A04

R程式設計

(含R程式碼風格指引)

吳漢銘

國立政治大學 統計學系

https://hmwu.idv.tw

本章大綱&學習目標

- 流程控制、程式流程圖:
  - 條件判別與執行: **`if else`**
  - 外顯迴圈: **`for, while, repeat`**
  - 迴圈的控制: **`next, break, switch`**
- 撰寫自訂函式: **`function()`**
- 隱含迴圈: **`apply, tapply, lapply, sapply`**
- 樣式比對: **`grep`**。搜尋與替換: **`sub, gsub, regexpr`**。**`which`**
- 集合運算、R程式執行時間、排序: **`Rank, Sort and Order`**
- 其它: Arguments 為函數、**`do.call`**、物件屬性強制轉換、查看指令程式碼
- R程式設計風格(Tidyverse Style Guide)及範例講解

Grouped Expression (Block)

```
{expr_1; ...; expr_m}
```

or

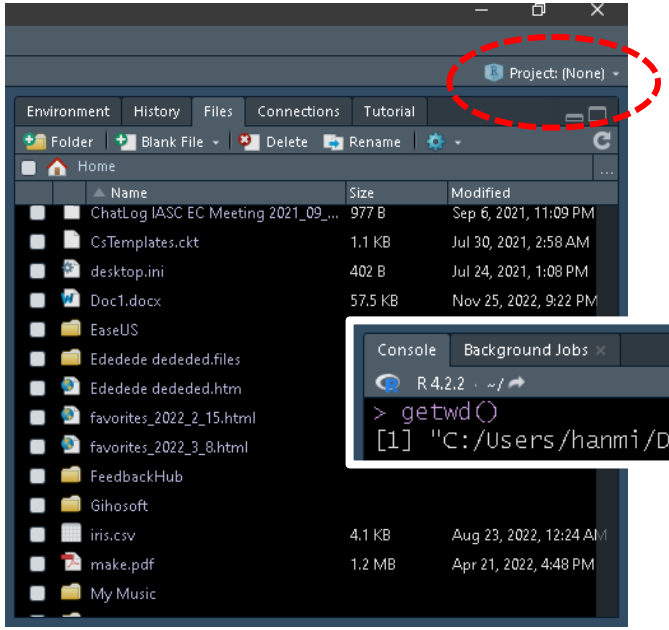
```
{
    expr_1
    ...
    expr_m
}
```

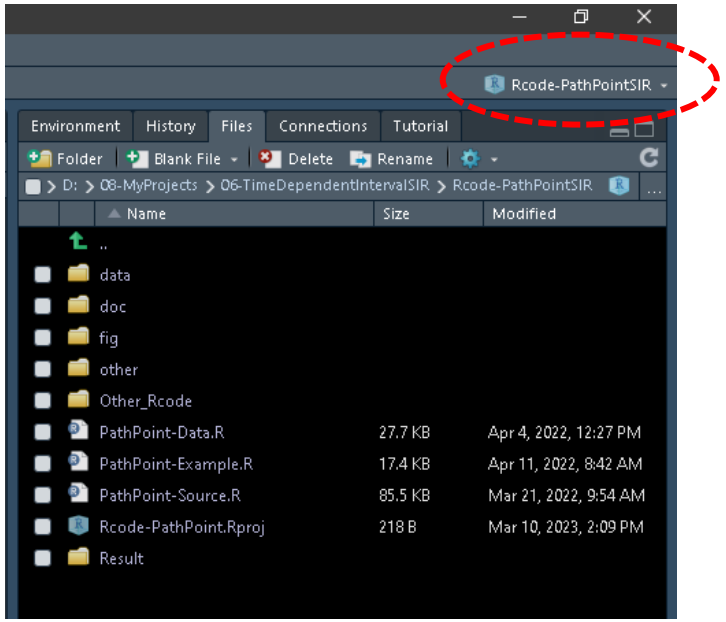
```
{ a <- c(1,2,3); b <- 5 }
```

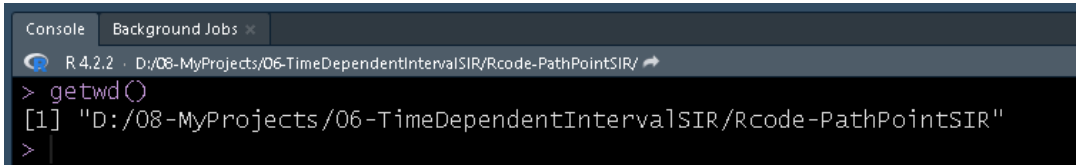
```
{
  a <- c(1,2,3)
  b <- 5
  c <- sum(a, b)
}
```

expression (表達句):
運算式可產生邏輯判斷結果 (x > 3)
statement (敘述句):
可單獨執行之完整運算 (x <- 3 + 5)

建議使用「R專案」工作。



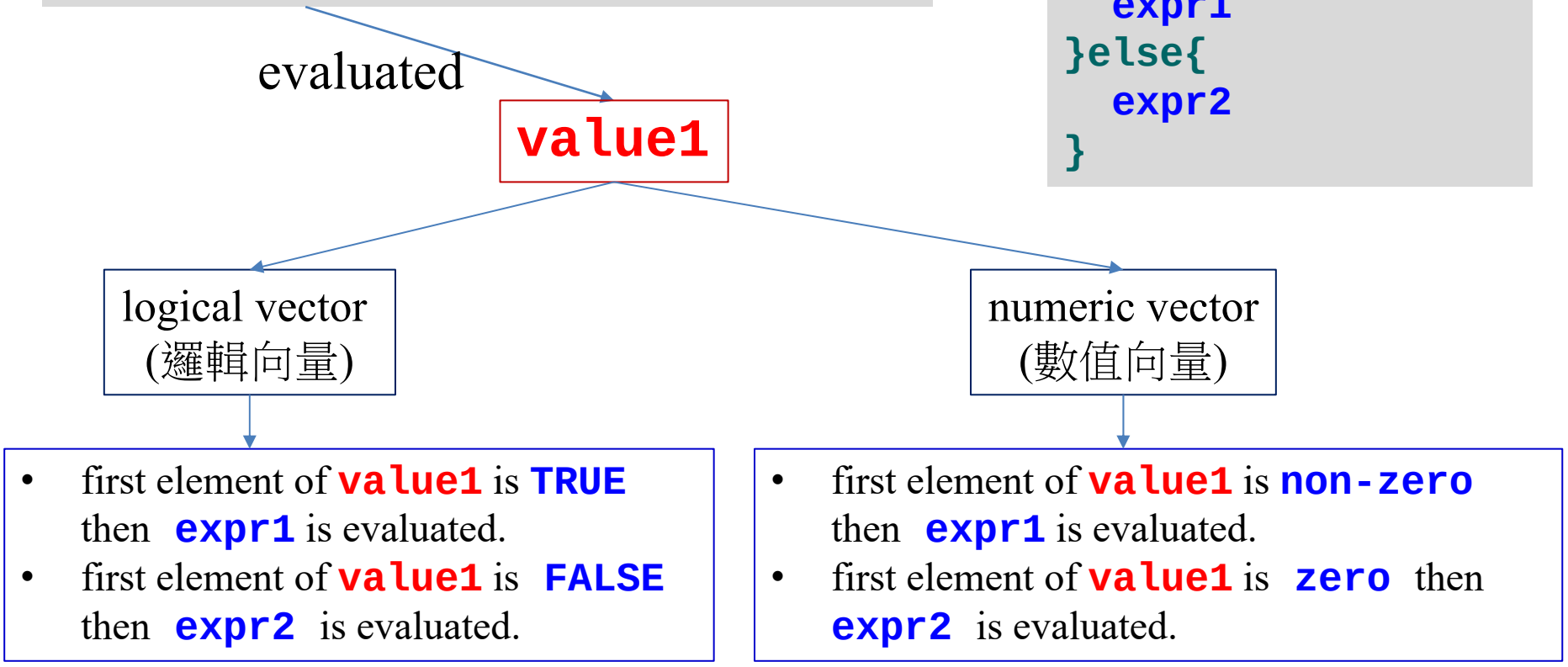




條件執行: if

```
if (condition) expr1 else expr2
```

```
if (condition){
  expr1
}else{
  expr2
}
```



- Only the **first element** of `value1` is used.
- If `value1` has any other type, an error is signaled.

課堂練習1: If value1 is a logical vector

```
> x <- 1
> if((x-2) < 0) cat("expr1 \n") else cat("expr2 \n")
expr1
>
> if((x-2) > 0) cat("expr1 \n") else cat("expr2 \n")
expr2
```

```
> x <- c(-1, 2, 3)
> if((x-2) < 0) cat("expr1 \n") else cat("expr2 \n")
expr1
Warning message:
In if ((x - 2) < 0) cat("expr1 \n") else cat("expr2 \n") :
  the condition has length > 1 and only the first element will be used

> if((x-2) > 0) cat("expr1 \n") else cat("expr2 \n")
expr2
Warning message:
In if ((x - 2) > 0) cat("expr1 \n") else cat("expr2 \n") :
  the condition has length > 1 and only the first element will be used
```

課堂練習2: If expr.1 is a numeric vector

```
> x <- 0
> if(x) cat("expr1 \n") else cat("expr2 \n")
expr2
> if(x+1) cat("expr1 \n") else cat("expr2 \n")
expr1
>
```

```
> x <- c(-1, 0, 1, 2,3)
> if(x) cat("expr1 \n") else cat("expr2 \n")
expr1
Warning message:
In if (x) cat("expr1 \n") else cat("expr2 \n") :
  the condition has length > 1 and only the first element will be used

> if(x+1) cat("expr1 \n") else cat("expr2 \n")
expr2
Warning message:
In if (x + 1) cat("expr1 \n") else cat("expr2 \n") :
  the condition has length > 1 and only the first element will be used
```

課堂練習3

```
> x <- c(-1, 2, 3)
> if(any(x <=0)) y <- log(1+x) else y <- log(x)
> y
[1]      -Inf 1.098612 1.386294
> z <- if(any(x<=0)) log(1+x) else log(x)
> z
[1]      -Inf 1.098612 1.386294
```

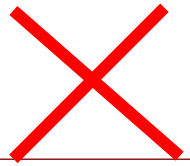
```
all() #return TRUE if all values are TRUE
any() #return TRUE if any values are TRUE
```

```
> x <- c(-1, 2, 3)
> if(any(x <=0)){
+     y <- log(1+x)
+ } else{
+    y <- log(x)
+ }
> y
[1]      -Inf 1.098612 1.386294
>
>
>
>
> x <- c(-1,2,3)
> if(any(x <=0)){
+     y <- log(1+x)
+ }
> else{
Error: unexpected 'else' in "else"
>    y <- log(x)
Warning message:
In log(x) : NaNs produced
> }
Error: unexpected '}' in "}"
> y
[1]       NaN 0.6931472 1.0986123
```

這種寫法比較好

(程式編輯器)

```
x <- c(-1,2,3)
if(any(x <=0)){
    y <- log(1+x)
} else{
   y <- log(x)
}
```



```
x <- c(-1,2,3)
if(any(x <=0)){
    y <- log(1+x)
}
else{
   y <- log(x)
}
```

比較兩數列是否一樣?

```
> (a <- sample(1:5))
[1] 1 5 4 3 2
> (b <- sample(1:5))
[1] 4 5 3 2 1
> a == b
[1] FALSE  TRUE FALSE FALSE FALSE
> any(a == b)
[1] TRUE
> all(a == b)
[1] FALSE
```

```
check.if <- function(a, b){
  if(a == b){
    cat("Equal! \n")
  }else{
    cat("Not equal! \n")
  }
}
```

```
check.if2 <- function(a, b){
  if(sum(abs(a - b)) == 0){
    cat("Equal! \n")
  }else{
    cat("Not equal! \n")
  }
}
```

```
> a <- sample(1:10, 4)
> b <- a
> check.if2(a, b)
Equal!
```

```
> check.if(a = 1, b = 1)
Equal!
> check.if(a = 1, b = 2)
Not equal!
> check.if(a = 1, b = c(1, 2, 3))
Equal!
Warning message:
In if (a == b) { : 條件的長度 > 1，因此只能用其第一元素
> check.if(a = 1, b = c(2, 1, 3))
Not equal!
Warning message:
In if (a == b) { : 條件的長度 > 1，因此只能用其第一元素
> check.if(a = c(1, 2, 3), b = c(1, 2, 3))
Equal!
Warning message:
In if (a == b) { : 條件的長度 > 1，因此只能用其第一元素
> check.if(a = c(2, 4, 5), b = c(1, 2, 3))
Not equal!
Warning message:
In if (a == b) { : 條件的長度 > 1，因此只能用其第一元素
> check.if(a = c(1, 5), b = c(4, 2, 3))
Not equal!
Warning messages:
1: In a == b : 較長的物件長度並非較短物件長度的倍數
2: In if (a == b) { : 條件的長度 > 1，因此只能用其第一元素
```

```
> identical(a, b) # identical {base}: Test Objects for Exact Equality
[1] TRUE
> all.equal(pi, 355/113) # all.equal {base}: Test if Two Objects are (Nearly) Equal
[1] "Mean relative difference: 8.491368e-08"
```

多個條件判斷: &、|、&&、||

- apply element-wise to vectors
  - &: #and
  - |: #or
- apply to vector
  - &&: #and
  - ||: #or

```
> a <- c(3, 4, 6, 9, 5)
> b <- c(1, 2, 8, 5, 6)
> a | b
[1] TRUE TRUE TRUE TRUE TRUE
> a & b
[1] TRUE TRUE TRUE TRUE TRUE
> a ||  b
[1] TRUE
> a && b
[1] TRUE
```

- 若運算對象是一個數字變數，則&&、|| 和 &、| 沒有差別。
- 使用&結合兩個條件，傳回真假值判別向量。
- 使用&&結合兩個條件，只傳回判別向量的第一個真假值元素。

- 在if內要「比較兩條件」時應採用"==" 而不是"="。

```
if(cond1 & cond2){
        ...
}

if(cond1 | cond2){
        ...
}

if(cond1 && cond2){
        ...
}

if(cond1 || cond2){
        ...
}

if(expr2 == expr1){
        ...
}
```

課堂練習4.1

```
> x <- 3
> y <- 4
>
> x < 2
[1] FALSE
> y > 2
[1] TRUE
> x < 2 || y > 2
[1] TRUE

> x > 2
[1] TRUE
> y > x
[1] TRUE
> x > 2 && y > x
[1] TRUE
```

```
> x < 2 | y > 2
[1] TRUE

> x > 2 & y > x
[1] TRUE
```

```
> xv <- c(1, 2, 3)
> yv <- c(2, 2, 5)

> xv < 2
[1] TRUE FALSE FALSE
> yv > 2
[1] FALSE FALSE  TRUE

> xv < 2 || yv > 2
[1] TRUE
> (! xv < 2) || yv > 2
[1] FALSE
> xv < 2 || (! yv > 2)
[1] TRUE

> xv < 2 && yv > 2
[1] FALSE
> (! xv < 2) && yv > 2
[1] FALSE
> xv < 2 && (! yv > 2)
[1] TRUE
```

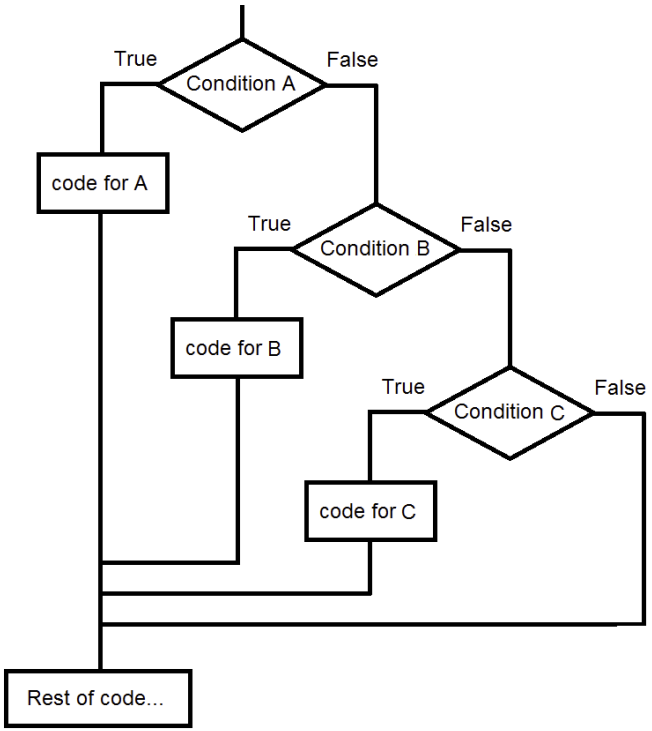
```
> xv < 2 | yv > 2
[1]  TRUE FALSE  TRUE
> (! xv < 2) | yv > 2
[1] FALSE  TRUE  TRUE
> xv < 2 | (! yv > 2)
[1]  TRUE  TRUE FALSE

> xv < 2 & yv > 2
[1] FALSE FALSE FALSE
> (! xv < 2) & yv > 2
[1] FALSE FALSE  TRUE
> xv < 2 & (! yv > 2)
[1]  TRUE FALSE FALSE
```

巢狀 if/else: Nested if/else

```
 if (test_expression1) {
   statement1
} else if (test_expression2) {
   statement2
} else if (test_expression3) {
   statement3
} else
   statement4
```

```
x <- 0
if (x < 0) {
   print("Negative number")
} else if (x > 0) {
   print("Positive number")
} else
   print("Zero")
```

```
a <- 2.13

if( a > 10 ){
    cat("a > 10 \n")
}else if(a > 5){
    cat("5 < a < 10 \n")
}else if(a > 2.5){
    cat("2.5 < a < 5 \n")
}else if(a > 1.25){
    cat("1.25 < a < 2.5 \n")
}else{
    cat("a < 1.25")
}

1.25 < a < 2.5
```



ifelse(condition, a ,b)

Return a vector of the length of its longest argument, with elements a[i] if condition[i] is true, otherwise b[i].

```
> (x <- c(2:-1))
[1]  2  1  0 -1
> sqrt(x)
[1] 1.414214 1.000000 0.000000      NaN
Warning message:
In sqrt(x) : NaNs produced
> sqrt(ifelse(x >= 0, x, NA))
[1] 1.414214 1.000000 0.000000       NA
> ifelse(x >= 0, sqrt(x), NA)
[1] 1.414214 1.000000 0.000000       NA
Warning message:
In sqrt(x) : NaNs produced
```

```
> (yes <- 5:6)
[1] 5 6
> (no <- pi^(0:2))
[1] 1.000000 3.141593 9.869604
> ifelse(NA, yes, no)
[1] NA
> ifelse(TRUE, yes, no)
[1] 5
> ifelse(FALSE, yes, no)
[1] 1
> ifelse(c(TRUE, F), yes, no)
[1] 5.000000 3.141593
```

```
ifelse(<condition>, <yes>, ifelse(<condition>, <yes>, <no>))
ifelse(<condition>, ifelse(<condition>, <yes>, <no>), <no>)
ifelse(<condition>,
       ifelse(<condition>, <yes>, <no>),
       ifelse(<condition>, <yes>, <no>)
      )
ifelse(<condition>, <yes>,
       ifelse(<condition>, <yes>,
              ifelse(<condition>, <yes>, <no>)
             )
       )
```

`ifelse()` can be nested in many ways:

課堂練習4.2

```
> x <- c(24, 13, 26, 21,  7,  9, 2, 1, 30, 14, 20, 16, 6, 4, 12, 8,
11, 22, 18, 3)
> ifelse(x <= 10, 1, ifelse(x <= 20, 2, 3))
 [1] 3 2 3 3 1 1 1 1 3 2 2 2 1 1 2 1 2 3 2 1
```

- 將年齡資料轉換為年齡群組1~20, 21~40, 41~60, 61歲以上，並編碼為A, B, C, D。

```
> set.seed(12345)
> age <- sample(1:100, 20)
> age
 [1] 73 87 75 86 44 16 31 48 67 91  4 14 65  1 34 40 33 97 15 78
```

  - 將”A”與”E”編碼為1，”C”編碼為2，”B”與”D”編碼為3。

```
> set.seed(12345)
> code <- sample(LETTERS[1:5], 20, replace=T)
> code
 [1] "D" "E" "D" "E" "C" "A" "B" "C" "D" "E" "A" "A" "D" "A" "B" "C"
[17] "B" "C" "A" "E"
```

提示: %in%

See also: `cut()`, `recode{car}`

課堂練習4.3

- 美國大學成績平均績點(GPA)(四分制)的計算方式如右表，請寫一R函式，將某同學之各科修課成績百分數score轉成等級及GPA。

等級 (Grade)	百分數	GPA
A	80 – 100 分	4
B	70 – 79 分	3
C	60 – 69 分	2
D	50 – 59 分	1
E	49 分以下	0

```
> set.seed(12345)
> score <- sample(0:100, 10, replace=T)
> score
 [1] 72 88 76 89 46 16 32 51 73 99
```

```
gpa.table <- data.frame(grade=c("A", "B", "C", "D", "E"),
                        pscore=c("80-100", "70-79", "60-69", "50-59", "49-0"),
                        GPA=c(4, 3, 2, 1, 0))
gpa.table
set.seed(12345)
score <- sample(0:100, 10, replace=T)

score_to_gpa <- function(x){

    group.id <- ifelse(x >= 80, 1,
                    ifelse(x >= 70, 2,
                           ifelse(x >=60, 3,
                                  ifelse(x >= 50, 4, 5))))
    data.frame(score=x, gpa.table[group.id,], row.names = NULL)
}
```

```
> score_to_gpa(score)
   score grade pscore GPA
1     72     B  70-79   3
2     88     A 80-100   4
3     76     B  70-79   3
4     89     A 80-100   4
5     46     E   49-0   0
6     16     E   49-0   0
7     32     E   49-0   0
8     51     D  50-59   1
9     73     B  70-79   3
10    99     A 80-100   4
```

程式設計的步驟

程式設計的主要目的，是利用電腦來解決問題。從面對問題，到程式設計完成，通常會經過六個階段：

1. **分析問題:** 探討以電腦解決問題的可行性、找出輸入輸出的資料項目等。
2. **找出演算法(Algorithm) :** 以電腦解決可能有許多種不同的方法(處理步驟)。每一種方法都是一個演算法。
3. **繪製流程圖或列出演算法步驟:**
   - 流程圖: 將處理問題的步驟，或一連串工作程序，用標準化的圖形和線條表現出來。
   - 可使用文字敘述的方式列出演算法步驟，來表達程式設計的思考邏輯，或程式的流程。
4. **撰寫程式:** 根據流程圖或演算法步驟撰寫程式。
5. **測試程式:** 反覆以多組輸入資料測試，以去除語法錯誤(Syntax Error)和邏輯錯誤(Logic Error)。
6. **編寫文件:** 註解(Remark)、說明文件、操作手冊等。

Source: http://163.20.173.51/vb/程式設計的步驟.htm

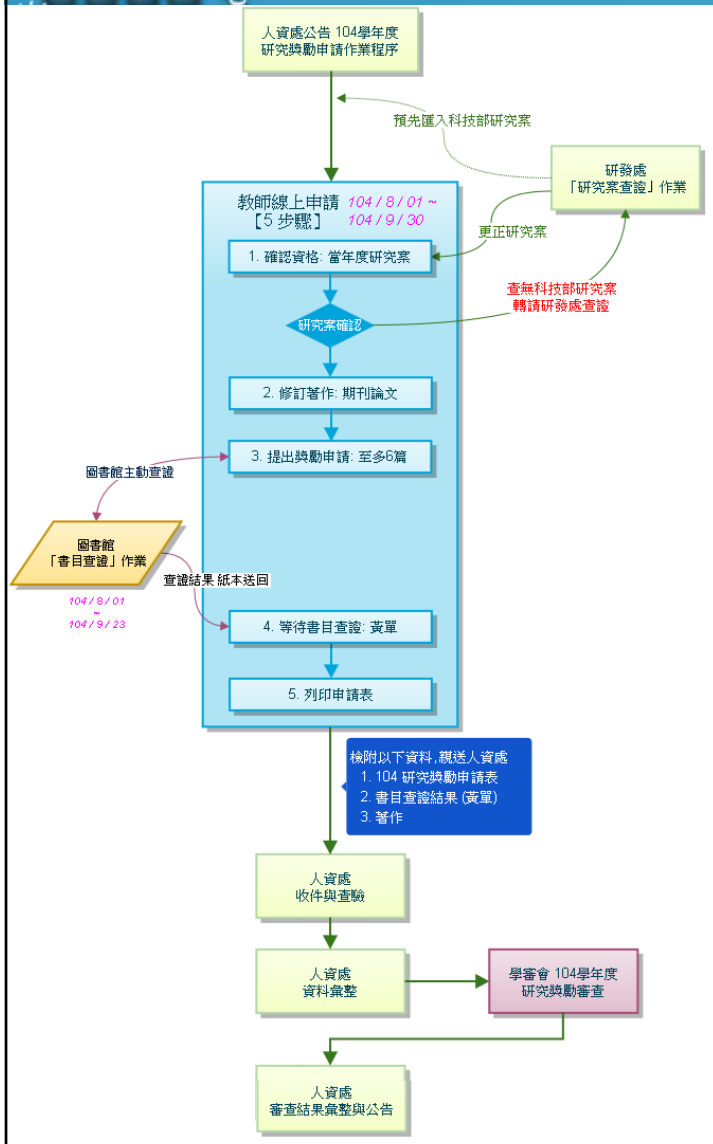
演算法和流程圖

- **演算法 (algorithm)：**
  - 由有限 (finite) 步驟 (step) 所構成的集合，依照給定輸入 (input) 依序執行每個明確 (definite) 且有效 (effective) 的步驟，以便能夠解決特定的問題；而步驟的執行必定會終止 (terminate)，並產生輸出 (output) 。

- **流程圖(flow chart):**
  - 就是利用各種方塊圖形、線條及箭頭等符號來表達問題的解決問題的步驟及進行的順序。
  - 流程圖是演算法的一種表示方式。
  - 一般而言，從這些符號本身的形狀，就可以看出記載資料的媒體，使用機器的種類、處理的方法及工作程序等特殊意義。
  - 在符號內也可以加入一些運算式或說明文字，增加它的可讀性。

Source: http://staff.csie.ncu.edu.tw/jrjiang/alg2014/book1-2.3+AB.pdf

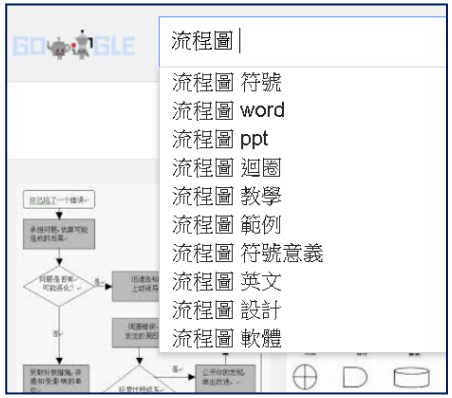
系統流程圖
(system flow chart)

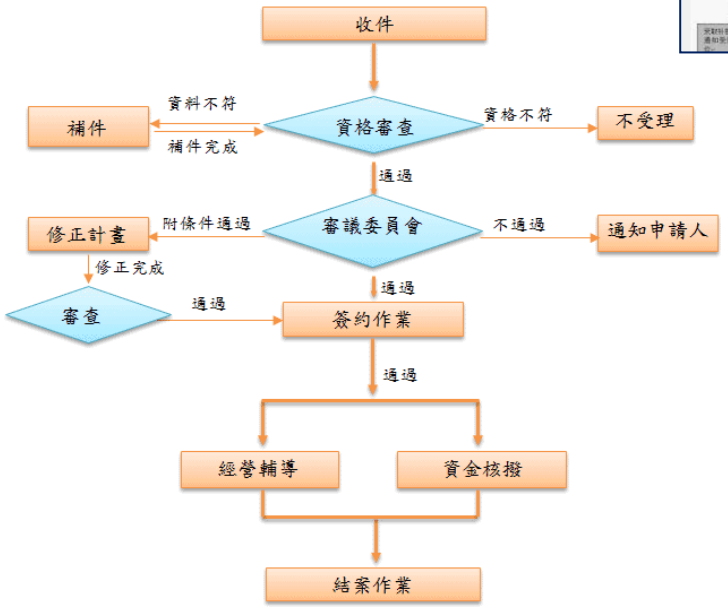


Source: http://award.tku.edu.tw/award-flowchart.cshtml

系統流程圖用以描述整個工作系統中，各單位之間的作業關係。

自行google「流程圖」





Source: 行政院國家發展基金管理會
國發基金創業天使計畫申請流程圖
http://www.angel885.org.tw/index.php?doc=apply04

程式流程圖
(program flow chart)

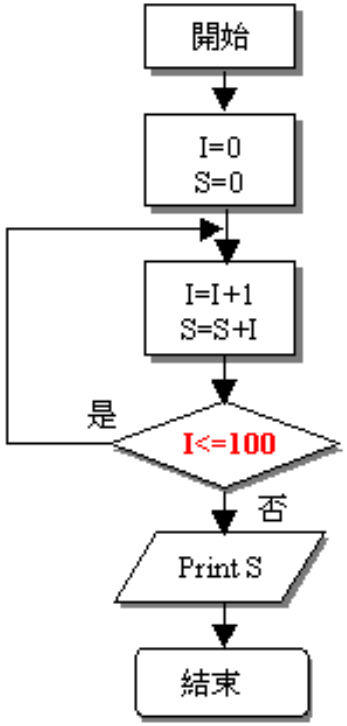
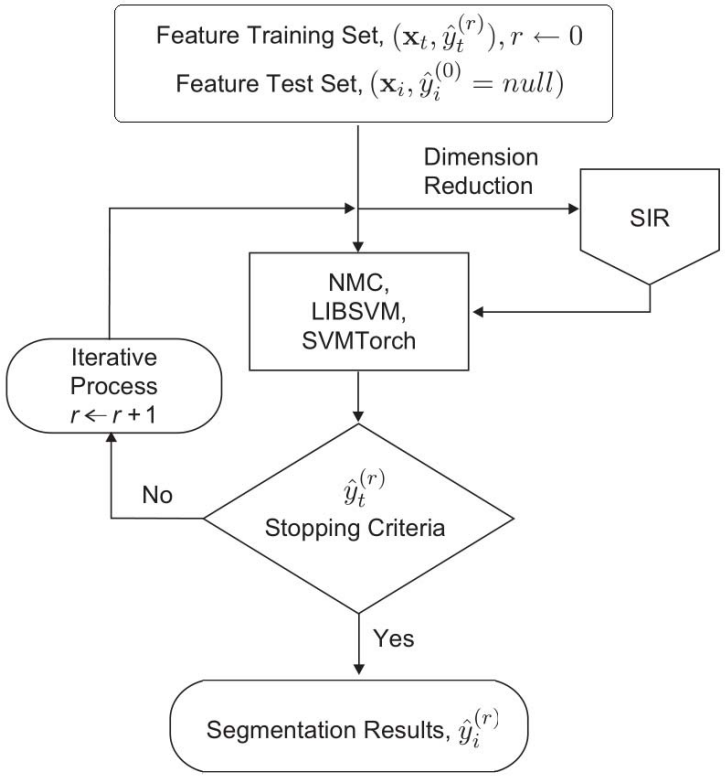


圖 3.4 求 1 加至 100 之間流程圖

程式流程圖用以表示程式中的處理過程。



Source: http://www.twivs.tnc.edu.tw/tht/chwa_bcc/book2/ch3/3-2.htm

一些程式流程圖範例:

http://www2.lssh.tp.edu.tw/~hlf/class-1/lang-c/flow/flow-chat.htm

演算法的設計

- 基本上，演算法的設計必須滿足下列準則：
  - **輸入資料**：明確指出程式中，要輸入哪些資料，如何輸入。
  - **輸出結果**：至少輸出一個以上的輸出結果。
  - **明確性**：所描述的程序是必須明確可行。
  - **有限性**：必須限定在有限數目的步驟內完成工作。
  - **有效性**：每一個步驟都可以用有效的指令表達出來。

- 一般而言，演算法不外乎三大步驟 ：

  (1) 輸入資料、

  (2) 處理資料、

  (3) 輸出結果。

撰寫自訂函式: function()

```
function_name <- function(input.var1, input.var2){
  output.var1 <- expr1
  command1
  ...
  output
}
```

```
my_dist <- function(x1, y1, x2, y2){
  d <- sqrt((x1-x2)^2 + (y1-y2)^2)
  d
}
```

```
function_name <- function(input.var1, input.var2 = default.value){
  output.var1 <- expr1
  command1
  ...
  output.var2 <- expr2
  list(output.name1 = output.var1, output.name2 = output.var2)
}
```

```
my_dist2 <- function(x1, y1, x2 = 0, y2 = 0){
  d <- sqrt((x1-x2)^2 + (y1-y2)^2)
  list(points.a = c(x1, y1), points.b = c(x2, y2), dist.ab = d)
}
```

執行函式

```
> function_name(input.var1, input.var2)
```

```
> my_dist(1, 2, 4, 7)
[1] 5.830952
```

```
> my_dist2(1, 2, 4, 7)
$points.a
[1] 1 2

$points.b
[1] 4 7

$dist.ab
[1] 5.830952
```

```
> my_dist2(1, 2)
$points.a
[1] 1 2

$points.b
[1] 0 0

$dist.ab
[1] 2.236068
```

引數及內定值

引數arguments: "name = object"

```
my_fun <- function(data, data.frame, is.graph, limit){ . . . }

# 執行
> my_fun(data = d, data.frame = df, is.graph = TRUE, limit = 20)
> my_fun(d, df, TRUE, 20)
> my_fun(d, df, is.graph = TRUE, limit = 20)
> my_fun(data = d, limit = 20, is.graph = TRUE, data.frame = df)
```

預設值 (Defaults)

```
my_fun <- function(data, data.grame, is.graph = TRUE, limit = 20){. . . }

# 執行
> ans <- my_fun(d, df)
> ans <- my_fun(d, df, limit = 10)
```

```
my_dist <- function(a, b){
   sqrt(sum((a-b)^2))
}
```

```
> my_dist(a = c(1, 2), b = c(4, 7))
[1] 5.830952
```

$$f(x) = x^2 + 1$$

```
> f <- function(x){
+   x^2+1
+ }
>
> x <- 1:5
> y <- f(x)
> y
[1]  2  5 10 17 26
```

```
> x <- 1:5
> y <- x^2+1
> y
[1]  2  5 10 17 26
```

函式之回傳值

```
> min(5:1, pi)
[1] 1
> pmin(5:1, pi)
[1] 3.141593 3.141593 3.000000 2.000000 1.000000
```

```
parmax <- function(a, b){
    c <- pmax(a,b)
    median(c)
}
> x <- c(1, 9, 2, 8, 3, 7)
> y <- c(9, 2, 8, 3, 7, 2)
> parmax(x,y)
[1] 8
```

```
data_ratio <- function(x){
    x.number <- length(x)
    x.up <- mean(x) + sd(x)
    x.down <- mean(x) - sd(x)
    x.n <- length(x[x.down < x & x < x.up])
    x.p <- x.n/x.number
    list(number = x.n, percent = x.p)
}


> data_ratio(iris[,1])
$number
[1] 90

$percent
[1] 0.6
```

課堂練習5

```
compute <- function(a, b=0.5){

    sum <- a + b
    diff <- a - b
    prod <- a * b
    if(b != 0){
      div <- a / b
    }else{
      div <- "divided by zero"
    }
    list(sum=sum, diff=diff, product=prod, divide=div)
}
```

```
> norm <- function(x) sqrt(x%*%x)
> norm(1:4)
         [,1]
[1,] 5.477226
```

```
> compute(2, 5)
$sum
[1] 7

$diff
[1] -3

$product
[1] 10

$divide
[1] 0.4
```

```
> compute(2)
$sum
[1] 2.5

$diff
[1] 1.5

$product
[1] 1

$divide
[1] 4
```

```
> compute(2, 0)
$sum
[1] 2

$diff
[1] 2

$product
[1] 0

$divide
[1] "divided by zero"
```

課堂練習6: 兩樣本之t檢定

```
two_sample_test <- function(y1, y2){
    n1 <- length(y1); n2 <- length(y2)
    m1 <- mean(y1); m2 <- mean(y2)
    s1 <- var(y1); s2 <- var(y2)
    s <- ((n1-1)*s1 + (n2-1)*s2)/(n1+n2-2)
    stat <- (m1-m2)/sqrt(s*(1/n1+1/n2))
    list(means=c(m1, m2), pool.var=s, stat=stat)
}
> t.stat <- two_sample_test(iris[,1], iris[,2])
> t.stat
$means
[1] 5.843333 3.057333

$pool.var
[1] 0.4378365

$stat
[1] 36.46328
```

Equal or unequal sample sizes, equal variance [edit]

This test is used only when it can be assumed that the two distributi violated, see below.) Note that the previous formulae are a special c = n_2. The t statistic to test whether the means are different can be ca

$$t = \frac{\bar{X}_1 - \bar{X}_2}{s_p \cdot \sqrt{\frac{1}{n_1} + \frac{1}{n_2}}}$$

where

$$s_p = \sqrt{\frac{(n_1 - 1)s_{X_1}^2 + (n_2 - 1)s_{X_2}^2}{n_1 + n_2 - 2}}.$$

is an estimator of the pooled standard deviation of the two samples:

```
> ?t.test
> t.test(iris[,1], iris[,2], var.equal = T)

        Two Sample t-test

data:  iris[, 1] and iris[, 2]
t = 36.463, df = 298, p-value < 2.2e-16
alternative hypothesis: true difference in means is not equal to 0
95 percent confidence interval:
 2.635637 2.936363
sample estimates:
mean of x mean of y
 5.843333  3.057333
```

函式內的變數

- Any ordinary assignments done within the function are local and temporary and are lost after exit from the function.

```
> rm(list=ls())
> my_sqrt_sum <- function(x, y){
    a <- sqrt(x)
    b <- sqrt(y)
    c <- a + b
    c
}

> a <- 4
> b <- 9
> my_sqrt_sum(a, b)
[1] 5
> a
[1] 4
> b
[1] 9
```

```
> rm(list=ls())
> my_sqrt_sum <- function(x, y){
    a <- sqrt(x)
    b <- sqrt(y)
    c <- a + b
    c
}
> my_sqrt_sum(4, 9)
[1] 5
> a
Error: object "a" not found
> b
Error: object "b" not found
```

函式內的變數

```
> rm(list=ls())
> y <- 9
> my_sqrt_sum <- function(x){
    a <- sqrt(x)
    b <- sqrt(y)
    y <- sqrt(y)
    c <- a + b
    c
}
> my_sqrt_sum(4)
[1] 5
> a
Error: object "a" not found
> b
Error: object "b" not found
> y
[1] 9
```

```
rm(list=ls())
Y.VALUE <- 9
my_sqrt_sum <- function(x){
    a <- sqrt(x)
    b <- sqrt(Y.VALUE)
    c <- a + b
    c
}
my_sqrt_sum(4)
[1] 5
```

```
rm(list=ls())
my_sqrt_sum <- function(x, y){
    x <- sqrt(x)
    y <- sqrt(y)
    c <- x + y
    c
}

>  x <- 4
>  y <- 9
>  x <- my_sqrt_sum(x, y)
>  x
[1] 5
>  y
[1] 9
```

課堂練習7.1: <<-

```
myfun1 <- function(x){
   y <- x + 5
   cat("y: ", y, "\n")
}

myfun2 <- function(x){
   y <<- x + 5
   cat("y: ", y, "\n")
}

y <- 5; cat("y: ", y, "\n")
myfun1(3)
cat("y: ", y, "\n")

y <- 5; cat("y: ", y, "\n")
myfun2(3)
cat("y: ", y, "\n")
```

```
> myfun1 <- function(x){
+    y <- x + 5
+    cat("y: ", y, "\n")
+ }
>
> myfun2 <- function(x){
+    y <<- x + 5
+    cat("y: ", y, "\n")
+ }
>
> y <- 5; cat("y: ", y, "\n")
y:  5
> myfun1(3)
y:  8
> cat("y: ", y, "\n")
y:  5
>
>
> y <- 5; cat("y: ", y, "\n")
y:  5
> myfun2(3)
y:  8
> cat("y: ", y, "\n")
y:  8
```

課堂練習7.2

計算數列x的個數，平均及標準差。

```
my_stat <- function(x){
    x.number <- length(x)
    x.mean <- mean(x)
    x.sd <- sd(x)
    list(number=x.number, mean=x.mean, sd=x.sd)
}

> my_stat(iris[,1])
$number
[1] 150

$mean
[1] 5.843333

$sd
[1] 0.8280661
```

'...' 參數

- Allows the function to accept additional arguments of unspecified name and number.
- If a function has '...' as a formal argument then any actual arguments that do not match a formal argument are matched with '...'.
- '...' is used in the argument list to specify that an arbitrary number of arguments are to be passed to the function.

```
> lm
function (formula, data, subset, weights, na.action, method = "qr",
    model = TRUE, x = FALSE, y = FALSE, qr = TRUE, singular.ok = TRUE,
    contrasts = NULL, offset, ...)
```

```
> myfun <- function(x, ...){
+     y <- mean(...) + x
+     y
+ }
>
> data <- rnorm(40)
> myfun(6, data)
[1] 5.997225
```

課堂練習7.3

Here is a function that takes any number of vectors and calculates their means and variances.

```
many_means <- function(...){

    #use [[]] subscripts in addressing its elements.
    data <- list(...)
    n <- length(data)
    means <- numeric(n)
    vars  <- numeric(n)
    for(i in 1:n){
        means[i] <-  mean(data[[i]])
         vars[i] <-  var(data[[i]])
    }

    print(means)
    print(vars)
}

> x <- rnorm(100); y <- rnorm(200); z <- rnorm(300)
> many_means(x,y,z)
[1] -0.007530678  0.031621030  0.026945631
[1] 0.8479211 0.9526169 1.1456980
```

課堂練習8

k為一常數，計算數列x在[mean(x)-k*sd(x), mean(x)+k*sd(x)]間的個數和比例。

```
data_k_ratio <- function(x, k=1){
   x.number <- length(x)
   x.mean <- mean(x)
   x.sd <- sd(x)
   x.up <- x.mean + k*x.sd;
   x.down <- x.mean - k*x.sd;
   x.n <- length(x[(x.down < x) & (x < x.up)])
   x.p <- x.n/x.number
   list(number=x.n, percent=x.p)
}
library(MASS)
data_k_ratio(drivers, 1)
data_k_ratio(drivers, 2)
data_k_ratio(drivers, 3)
```

```
> library(MASS)
> data_k_ratio(drivers, 1)
$number
[1] 134

$percent
[1] 0.6979167

> data_k_ratio(drivers, 2)
$number
[1] 185

$percent
[1] 0.9635417

> data_k_ratio(drivers, 3)
$number
[1] 191

$percent
[1] 0.9947917
```

迴圈 (Looping)

- 外顯迴圈(Explicit looping):
  `for, while, repeat`

- 隱含迴圈(Implicit looping):
  `apply, tapply, lapply, sapply`

for 迴圈

```
> for (name in expr_1) expr.2
```

- **name**: loop variable.
- **expr.1**: can be either a vector or a list.
- for each element in **expr.1** the variable name is set to the value of that element and **expr.2** is evaluated.

執行「多次有規律性的指令」

```
for(i in 1:5){
    cat("loop: ", i, "\n")
}
```

```
loop:  1
loop:  2
loop:  3
loop:  4
loop:  5
```

```
for(k in c(1, 17, 3, 56, 2)){
    cat(k, "\t")
}

for(bloodType in c("A", "AB", "B", "O")){
    cat(bloodType, "\t")
}
```

for 迴圈

```
rm(list=ls())
y <- round(rnorm(10), 2)
z <- y
y
i
for(i in 1:length(y)){
        if(y[i] < 0)
        y[i] <-0
}
y
i

z[z < 0] <- 0
z
```

- side effect: the variable name still exists after the loop has concluded and it has the value of the last element of vector that the loop was evaluated for.

```
> rm(list=ls())
> y <- round(rnorm(10), 2)
> z <- y
> y
 [1]  1.04  1.74 -0.05 -0.44 -0.71 -0.57  0.11 -0.06  0.32 -0.76
> i
Error: object "i" not found
> for(i in 1:length(y)){
+ if(y[i] < 0)
+ y[i] <-0
+ }
> y
 [1] 1.04 1.74 0.00 0.00 0.00 0.00 0.11 0.00 0.32 0.00
> i
[1] 10
>
> z[z < 0] <- 0
> z
 [1] 1.04 1.74 0.00 0.00 0.00 0.00 0.11 0.00 0.32 0.00
```

for雙迴圈

- 單一迴圈

```
a <- numeric(5)
for(i in 1:5){
  a[i]<- i^2
}
> a
[1]  1  4  9 16 25
```

- 雙迴圈

```
m <- 3
n <- 4
for(i in 1:m){
   for(j in 1:n){
     cat("loop: (", i, ",", j, ")\n")
   }
}
```

```
loop: ( 1 , 1 )
loop: ( 1 , 2 )
loop: ( 1 , 3 )
loop: ( 1 , 4 )
loop: ( 2 , 1 )
loop: ( 2 , 2 )
loop: ( 2 , 3 )
loop: ( 2 , 4 )
loop: ( 3 , 1 )
loop: ( 3 , 2 )
loop: ( 3 , 3 )
loop: ( 3 , 4 )
```

```
a <- matrix(0,2,4)
for(i in 1:2){
    for(j in 1:4){
       a[i,j]<- i+j
    }
}
> a
     [,1] [,2] [,3] [,4]
[1,]    2    3    4    5
[2,]    3    4    5    6
```

NOTE: 寫R程式，應儘量避免使用for雙迴圈。

迴圈的控制: next

- **next**: immediately causes control to return to the start of the loop.
  - The next iteration of the loop is then executed.
  - No statement below **next** in the current loop is evaluated.

```
m <- 3
n <- 4
for(i in 1:m){
   for(j in 1:n){

     if(i==2){
       cat("before next:", i,",",j, "\n")
       next
       cat("after next:", i,",",j, "\n")
     }else{
       cat("loop: (", i, ",", j, ")\n")
     }
   }
}
```

```
loop: ( 1 , 1 )
loop: ( 1 , 2 )
loop: ( 1 , 3 )
loop: ( 1 , 4 )
before next: 2 , 1
before next: 2 , 2
before next: 2 , 3
before next: 2 , 4
loop: ( 3 , 1 )
loop: ( 3 , 2 )
loop: ( 3 , 3 )
loop: ( 3 , 4 )
```

迴圈的控制: break

- **break**: causes an exit from the innermost loop that is currently being executed.

```
m <- 3
n <- 4
for(i in 1:m){
   for(j in 1:n){

     if(i==2){
       cat("before break:", i,",",j, "\n")
       break
       cat("after break:", i,",",j, "\n")
     }else{
       cat("loop: (", i, ",", j, ")\n")
     }
   }
}
```

```
loop: ( 1 , 1 )
loop: ( 1 , 2 )
loop: ( 1 , 3 )
loop: ( 1 , 4 )
before break: 2 , 1
loop: ( 3 , 1 )
loop: ( 3 , 2 )
loop: ( 3 , 3 )
loop: ( 3 , 4 )
```

課堂練習: 判斷一正整數是否為質數

```
check_prime <- function(num){

  yes <- FALSE

  if(num == 2){
    yes <- TRUE

  } else if(num > 2) {

    yes <- TRUE
    for(i in 2:(num-1)) {
        if ((num %% i) == 0) {
            yes <- FALSE
            break
        }
    }
  }

  if(yes) {
    cat(num, "is a prime number. \n")
  } else {
    cat(num, "is not a prime number. \n")
  }
}
```

```
> check_prime(2)
2 is a prime number.
> check_prime(13)
13 is a prime number.
> check_prime(25)
25 is not a prime number.
```

repeat and while

```
> repeat{expr.1}
```

- repeat: causes repeated evaluation of the body until a break is specifically requested.

```
> while(condition) expr.1
```

- condition is evaluated and if its values is TRUE than expr.1 is evaluated.
- This process continues until expr.1 evaluates to FALSE.
- If expr.1 is never evaluated then while returns NULL and otherwise it returns the value of the last evaluation of expr.1.

課堂練習9

```
a <- 5
while(a > 0){
    a <- a - 1
    cat(a,"\n")
    if(a == 2){
        cat("before next:", a, "\n")
        next
        cat("after next:", a, "\n")
    }
}
```

```
4
3
2
before next: 2
1
0
```

```
a <- 5
while(a > 0){

    if(a == 2){
        cat("before break:", a, "\n")
        break
    }
    a <- a - 1
    cat(a,"\n")
}
```

```
4
3
2
before break: 2
```

```
a <- 5
while(a > 0){

    if(a == 2){
        cat("before break:", a, "\n")
        next
        cat("after break:", a, "\n")
    }
    a <- a - 1
    cat(a,"\n")
}
```

無窮迴圈

停止執行: 按「Esc」或「Ctrl + c」或「Ctrl + z」

課堂練習10：計算n!

```
factorial_for <- function(n){
    f <- 1
    if(n < 2) return(1)
    for(i in 2:n){
        f <- f * i
    }
    f
}
factorial_for(5)
```

```
factorial_repeat <- function(n){
    f <- 1
    t <- n
    repeat{
        if(t < 2) break
        f <- f * t
        t <- t - 1
    }
    return(f)
}
factorial_repeat(5)
```

```
factorial_while <- function(n){
    f <- 1
    t <- n
    while(t > 1){
        f <- f * t
        t <- t - 1
    }
    return(f)
}
factorial_while(5)
```

```
factorial_call <- function(n, f){

    if(n <= 1){
      return(f)
    }
    else{
      factorial_call(n - 1, n * f)
    }
}
factorial_call(5, 1)
```

```
factorial_cumprod <- function(n) max(cumprod(1:n))
factorial_cumprod(5)
factorial(5)
```

switch(expr.1, list)

- **expr.1** is evaluated and the result value obtained.
- If value is a number between 1 and the length of list then the corresponding element list is evaluated and the result returned.
- If value is too large or too small **NULL** is returned.
- If there is no match **NULL** is returned.

```
> x <- 3
> switch(x,
         cat("2+2\n"),
         cat("mean(1:10)\n"),
         cat("sd(1:10)\n"))
sd(1:10)
> switch(x, 2+2, mean(1:10), sd(1:10))
[1] 3.027650
> switch(2, 2+2, mean(1:10), sd(1:10))
[1] 5.5
> switch(6, 2+2, mean(1:10), sd(1:10))
NULL
```

```
my_lunch <- function(y){
    switch(y,
           fruit="banana",
           vegetable="broccoli",
           meat="beef")
}
> my_lunch("fruit")
[1] "banana"
> my_lunch(fruit)
Error in switch(y, fruit = "banana",
vegetable = "broccoli", meat = "beef") :
  object "fruit" not found
```

課堂練習11: 計算中心程度

```
x_center <-  function(x, type){
    switch(type,
           mean = mean(x),
           median = median(x),
           trimmed = mean(x, trim = 0.1),
           stop("Measure is not included!"))
}

> x <- rnorm(20)
> x_center(x, "mean")
[1] 0.1086806
> x_center(x, "median")
[1] 0.2885969
> x_center(x, "trimmed")
[1] 0.2307617
> x_center(x, "mode")
Error in switch(type, mean = mean(x), median = median(x), trimmed
= mean(x,  :
  Measure is not included!
```

課堂練習12: 計算median

```
my_median_1 <- function(x){
    odd.even <- length(x)%%2
    if(odd.even == 0){
        (sort(x)[length(x)/2] + sort(x)[1+length(x)/2])/2
    }else{
        sort(x)[ceiling(length(x)/2)]
    }
}
```

```
my_median_2 <- function(x){
    odd.even <- length(x)%%2
    s.x <- sort(x)
    n <- length(x)
    if(odd.even == 0){
        median <- (s.x[n/2] + s.x[1+n/2])/2
    }else{
        median <- s.x[ceiling(n/2)]
    }
    return(median)
}
```

```
> x <- rnorm(30)
> my_median_1(x)
[1] -0.06110589
> my_median_2(x)
[1] -0.06110589
> median(x)
[1] -0.06110589
```

apply : Apply Functions Over Array Margins

```
> (x <- matrix(1:24, nrow=4))
     [,1] [,2] [,3] [,4] [,5] [,6]
[1,]    1    5    9   13   17   21
[2,]    2    6   10   14   18   22
[3,]    3    7   11   15   19   23
[4,]    4    8   12   16   20   24
>
> #1: rows, 2:columns
> apply(x, 1, sum)
[1] 66 72 78 84
> apply(x, 2, sum)
[1] 10 26 42 58 74 90
>
> #apply function to the individual elements
>
> apply(x, 1, sqrt)
         [,1]     [,2]     [,3]     [,4]
[1,] 1.000000 1.414214 1.732051 2.000000
[2,] 2.236068 2.449490 2.645751 2.828427
[3,] 3.000000 3.162278 3.316625 3.464102
[4,] 3.605551 3.741657 3.872983 4.000000
[5,] 4.123106 4.242641 4.358899 4.472136
[6,] 4.582576 4.690416 4.795832 4.898979
> apply(x, 2, sqrt)
         [,1]     [,2]     [,3]     [,4]     [,5]     [,6]
[1,] 1.000000 2.236068 3.000000 3.605551 4.123106 4.582576
[2,] 1.414214 2.449490 3.162278 3.741657 4.242641 4.690416
[3,] 1.732051 2.645751 3.316625 3.872983 4.358899 4.795832
[4,] 2.000000 2.828427 3.464102 4.000000 4.472136 4.898979
```

apply {base}: Apply Functions Over Array Margins

Description: Returns a vector or array or list of values obtained by applying a function to margins of an array or matrix.

Usage: `apply(X, MARGIN, FUN, ...)`

apply自定函式

將某班三科成績，皆以開根號乘以10重新計分。

```
> # generate score data
> math <- sample(1:100, 50, replace=T)
> english <- sample(1:100, 50, replace=T)
> algebra <- sample(1:100, 50, replace=T)
> ScoreData <- cbind(math, english, algebra)
> head(ScoreData, 5)
     math english algebra
[1,]    7      52      93
[2,]    7      17       9
[3,]   57      89      69
[4,]   69      21      97
[5,]   20      64      64
>
> myfun <- function(x){
+     sqrt(x)*10
+ }
> sdata1 <- apply(ScoreData, 2, myfun)
> head(sdata1, 5)
         math  english  algebra
[1,] 26.45751 72.11103 96.43651
[2,] 26.45751 41.23106 30.00000
[3,] 75.49834 94.33981 83.06624
[4,] 83.06624 45.82576 98.48858
[5,] 44.72136 80.00000 80.00000
```

```
> head(apply(ScoreData, 2, function(x) sqrt(x)*10), 5)
         math  english  algebra
[1,] 26.45751 72.11103 96.43651
[2,] 26.45751 41.23106 30.00000
[3,] 75.49834 94.33981 83.06624
[4,] 83.06624 45.82576 98.48858
[5,] 44.72136 80.00000 80.00000
>
> myfun2 <- function(x, attend){
+     y <- sqrt(x)*10 + attend
+     ifelse(y > 100, 100, y)
+ }
> sdata2 <- apply(ScoreData, 2, myfun2, attend=5)
> head(sdata2, 5)
         math  english   algebra
[1,] 31.45751 77.11103 100.00000
[2,] 31.45751 46.23106  35.00000
[3,] 80.49834 99.33981  88.06624
[4,] 88.06624 50.82576 100.00000
[5,] 49.72136 85.00000  85.00000
```

tapply : Apply a Function Over a "Ragged" Array

tapply {base}: Apply a Function Over a Ragged Array

Description: Apply a function to each cell of a ragged array, that is to each (non-empty) group of values given by a unique combination of the levels of certain factors.

Usage: `tapply(X, INDEX, FUN = NULL, ..., simplify = TRUE)`

```
> tapply(iris$Sepal.Width, iris$Species, mean)
    setosa versicolor  virginica
     3.428      2.770      2.974
```

```
> set.seed(12345)
> scores <- sample(0:100, 50, replace=T)
> grade <- as.factor(sample(c("大一", "大二", "大三", "大四"), 50, replace=T))
> bloodtype <- as.factor(sample(c("A","AB","B","O"), 50, replace=T))
> tapply(scores, grade, mean)
    大一     大二     大三     大四
51.69231 55.87500 35.06667 59.42857
> tapply(scores, bloodtype, mean)
       A       AB        B        O
68.88889 43.12500 54.18750 37.94118
> tapply(scores, list(grade, bloodtype), mean)
         A       AB    B        O
大一 96.00       NA 65.5 31.14286
大二 97.00 50.33333 71.0 42.66667
大三 47.25 13.00000 39.0 25.66667
大四 71.00 56.00000 60.0 55.50000
```

tapply : Apply a Function Over a "Ragged" Array

```
> n <- 20
> (my.factor <- factor(rep(1:3, length = n), levels = 1:5))
 [1] 1 2 3 1 2 3 1 2 3 1 2 3 1 2 3 1 2 3 1 2
Levels: 1 2 3 4 5
> table(my.factor)
my.factor
1 2 3 4 5
7 7 6 0 0
> tapply(1:n, my.factor, sum)
 1  2  3  4  5
70 77 63 NA NA
```

```
> tapply(1:n, my.factor, range)
$`1`
[1]  1 19
$`2`
[1]  2 20
$`3`
[1]  3 18
$`4`
NULL
$`5`
NULL
```

```
> tapply(1:n, my.factor, quantile)
$`1`
  0%  25%  50%  75% 100%
 1.0  5.5 10.0 14.5 19.0

$`2`
  0%  25%  50%  75% 100%
 2.0  6.5 11.0 15.5 20.0

$`3`
   0%   25%   50%   75%  100%
 3.00  6.75 10.50 14.25 18.00

$`4`
NULL

$`5`
NULL
```

```
# not run
# > by(iris[,1:4] , iris$Species , mean)
by(iris[,1:4] , iris$Species , colMeans)

varMean <- function(x, ...) sapply(x, mean, ...)
by(iris[, 1:4], iris$Species, varMean)
```

lapply : Apply a Function over a List or Vector

lapply returns a list of the same length as X, each element of which is the result of applying FUN to the corresponding element of X.

```
> a <- c("a", "b", "c", "d")
> b <- c(1, 2, 3, 4, 4, 3, 2, 1)
> c <- c(T, T, F)
>
> list.object <- list(a,b,c)
>
> my.la1 <- lapply(list.object, length)
> my.la1
[[1]]
[1] 4

[[2]]
[1] 8

[[3]]
[1] 3
```

```
> my.la2 <- lapply(list.object, class)
> my.la2
[[1]]
[1] "character"

[[2]]
[1] "numeric"

[[3]]
[1] "logical"
```

replicate: repeated evaluation of an expression

- **replicate** is a wrapper for the common use of sapply for repeated evaluation of an expression (which will usually involve random number generation).

```
replicate(n, expr, simplify = "array")
```

```
> rep(5.6, 3)
[1] 5.6 5.6 5.6
> replicate(3, 5.6)
[1] 5.6 5.6 5.6
> rep(rnorm(1), 3)
[1] 1.025571 1.025571 1.025571
> replicate(3, rnorm(1))
[1] -0.2847730 -1.2207177  0.1813035
> replicate(3, mean(rnorm(10)))
[1]  0.1843254  0.6546170 -0.5903897
>
> # toss two dices five times,
> # output the sum each time
> dice1 <- sample(1:6, 1)
> dice2 <- sample(1:6, 1)
> dice1 + dice2
[1] 11
```

```
> my_dice <- function(n){
+   dice.no <- sample(1:6, n, replace=T)
+   dice.sum <- sum(dice.no)
+   output <- c(dice.no, dice.sum)
+   names(output) <- c(paste0("dice", 1:n), "sum")
+   output
+ }
> my_dice(3)
dice1 dice2 dice3   sum
    2     5     3    10
> replicate(5, my_dice(2))
      [,1] [,2] [,3] [,4] [,5]
dice1    5    3    4    6    3
dice2    1    6    6    4    6
sum      6    9   10   10    9
```

Apply a Function over ....

- `sapply(X, FUN, ..., simplify = TRUE, USE.NAMES = TRUE)`
  - a user-friendly version of lapply by default returning a vector or matrix if appropriate.
- `mapply(FUN, ..., MoreArgs = NULL, SIMPLIFY = TRUE, USE.NAMES = TRUE)`
  - for applying a function to multiple arguments.
- `rapply(object, f, classes = "ANY", deflt = NULL, how = c("unlist", "replace", "list"), ...)`
  - for a recursive version of `lapply()`.
- `eapply(env, FUN, ..., all.names = FALSE, USE.NAMES = TRUE)`
  - for applying a function to each entry in an environment.
- `replicate(n, expr, simplify = "array")`
  - replicate is a wrapper for the common use of sapply for repeated evaluation of an expression (which will usually involve random number generation).
- `aggregate(x, ...)`
  - Splits the data into subsets, computes summary statistics for each, and returns the result in a convenient form.

See also: B01-1-hmwu_R-DataManipulation.pdf
`scale {base}`
Scaling and Centering of Matrix-like Objects
sweep which allows centering (and scaling) with arbitrary statistics.

課堂練習13

```
> (select.num <- sapply(iris, is.numeric))  #return vector
Sepal.Length  Sepal.Width Petal.Length  Petal.Width      Species
        TRUE         TRUE         TRUE         TRUE        FALSE
> iris[1:2, select.num]
  Sepal.Length Sepal.Width Petal.Length Petal.Width
1          5.1         3.5          1.4         0.2
2          4.9         3.0          1.4         0.2
> select.fac <- sapply(iris, is.factor)
> select.fac
Sepal.Length  Sepal.Width Petal.Length  Petal.Width      Species
       FALSE        FALSE        FALSE        FALSE         TRUE
> iris[1:5, select.fac]
[1] setosa setosa setosa setosa setosa
Levels: setosa versicolor virginica

> # don't use apply(iris, 2, is.numeric)
> apply(iris, 2, is.numeric)
Sepal.Length  Sepal.Width Petal.Length  Petal.Width      Species
       FALSE        FALSE        FALSE        FALSE        FALSE

> unique(iris$Species)
[1] setosa     versicolor virginica
Levels: setosa versicolor virginica
> table(iris$Species)
    setosa versicolor  virginica
        50         50         50
```

樣式比對: Pattern Matching

```
> wf <- read.table("worldfloras.txt", header=TRUE)
> attach(wf)
> names(wf)
> dim(wf)
[1] 161   7
```

Data Editor

	Country	Latitude	Area	Population	Flora	Endemism	Continent
1	Afghanistan	30	636	14.3	3000	0.27	Asia
2	Albania	42	29	3	3200	0.008	Europe
3	Algeria	35	2382	21.3	3139	0.08	N.Africa
4	Andorra	42	0.5	0.034	1000	0	Europe
5	Angola	25	1247	8.5	5000	0.25	Africa
6	Antarctica	85	14000	0	2	0	Antarctica
7	Argentina	45	2777	30.1	9000	0.27	S.America
8	Australia	25	7682	15.5	23000	0.8	Australia
9	Austria	48	84	7.5	3000	0.012	Europe
10	Bahrain	26	0.66	0.4	175	0	Asia
11	Balearic Islands	40	5	0.62	1400	0.067	Europe
12	Bangladesh	23	144	98.5	5000	-1	Asia
13	Belgium	52	31	9.9	1700	0	Europe
14	Belize	14	23	0.16	3240	0.046	C.America
15	Benin	5	113	3.9	2000	0.006	Africa
16	Bhutan	30	47	1.4	5000	0.12	Asia
17	Bolivia	16	1099	6.2	16500	0.23	S.America
18	Botswana	25	575	1	2015	0.008	Africa
19	Brazil	10	8512	132.6	55000	0.5	S.America
20	Brunei	5	5.7	0.27	6000	-1	SE.Asia
21	Bulgaria	42	111	9.2	3600	0.015	Europe
22	Burkina Faso	15	274	6.8	1096	0	Africa
23	Burma	20	678	38.5	7000	0.153	SE.Asia
24	Burundi	0	28	4.5	2500	0.04	Africa
25	Cameroon	5	475	9.5	9000	0.017	Africa
26	Canada	55	9922	25.3	3220	0.045	N.America
27	Central African Republic	0	625	2.5	3600	0.025	Africa
28	Chad	10	1284	4.9	1600	-1	Africa
29	Chile	40	752	11.9	5500	0.5	S.America
30	China	35	9597	1051.6	30000	0.1	Asia
31	Colombia	5	1139	28.1	45000	0.05	S.America
32	Congo	0	342	1.7	4000	0.2	Africa
33	Corsica	42	8.7	0.23	2200	0.014	Europe
34	Costa Rica	12	51	2.5	8000	0.17	C.America
35	Crete	35	8.3	0.46	1700	0.09	Europe
36	Cuba	20	114	10	7000	0.5	C.America
37	Cyprus	35	9.2	0.7	2000	0.058	Europe

「字串處理」**Ebook: Handling and Processing Strings in R**

http://gastonsanchez.com/resources/how-to/2013/09/22/Handling-and-Processing-Strings-in-R/

樣式比對: grep

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

- **ignore.case=FALSE**: the pattern matching is case sensitive
- **value=FALSE**: return integer indices of the matches. **value=TRUE**: return matching elements themselves.
- **invert=TRUE**: return indices or values for elements that do not match.

Select subsets of countries on the basis of specified patterns.

```
> index <- grep("R", as.character(Country)) # contain "R"
[1]  27  34  40 116 118 119 120 152
> as.vector(Country[index])
[1] "Central African Republic" "Costa Rica"              "Dominican Republic"
[4] "Puerto Rico"              "Reunion"                 "Romania"
[7] "Rwanda"                   "USSR"
> as.vector(Country[grep("^R", as.character(Country))]) # begin with "R"
[1] "Reunion" "Romania" "Rwanda"
> as.vector(Country[grep(" R", as.character(Country))]) # " R" with multiple name
[1] "Central African Republic" "Costa Rica"   "Dominican Republic"
[4] "Puerto Rico"
> as.vector(Country[grep(" ", as.character(Country))]) # two or more names
 [1] "Balearic Islands"         "Burkina Faso"             "Central African Republic"
...
[25] "Yemen North"              "Yemen South"
> as.vector(Country[grep("y$", as.character(Country))]) # ending by "y"
[1] "Hungary"  "Italy"    "Norway"   "Paraguay" "Sicily"   "Turkey"   "Uruguay"
```

樣式比對: grep

```
# select countries with names containing C to E
> my.pattern <- "[C-E]"
> index <- grep(my.pattern, as.character(Country))
> as.vector(Country[index])
 [1] "Cameroon"                "Canada"                  "Central African Republic"
...
[22] "Ivory Coast"             "New Caledonia"           "Tristan da Cunha"
# select countries with names containing C to E in the first
> as.vector(Country[grep("^[C-E]", as.character(Country))])
 [1] "Cameroon"                "Canada"                  "Central African Republic"
...
[19] "El Salvador"             "Ethiopia"
# select countries that do not end with a letter between ‘a’ and ‘t’.
> as.vector(Country[-grep("[a-t]$", as.character(Country))])
 [1] "Hungary"  "Italy"    "Norway"   "Paraguay" "Peru"     "Sicily"   "Turkey"   "Uruguay"
 [9] "USA"      "USSR"     "Vanuatu"
# select countries that do not end with a letter between ‘a’’A’ and ‘t’’T’.
> as.vector(Country[-grep("[A-T a-t]$", as.character(Country))])
[1] "Hungary"  "Italy"    "Norway"   "Paraguay" "Peru"     "Sicily"   "Turkey"   "Uruguay"
[9] "Vanuatu"
```

樣式比對: grep

'.' means anything

See also:
- strtrim{base}, substr{base}, substring{base}
- strsplit{base}

y is the second character
```
> as.vector(Country[grep("^.y", as.character(Country))])
[1] "Cyprus" "Syria"
```

y is the third character
```
> as.vector(Country[grep("^..y", as.character(Country))])
[1] "Egypt"     "Guyana"    "Seychelles"
```

y is the sixth character
```
> as.vector(Country[grep("^.{5}y", as.character(Country))])
[1] "Norway" "Sicily" "Turkey"
```

{,4} means 'repeat up to four' anything before $
```
> as.vector(Country[grep("^.{,4}$", as.character(Country))])
 [1] "Benin" "Burma" "Chad"  "Chile" "China" "Congo" "Crete" "Cuba"  "Egypt" "Gabon" "Ghana" "Haiti"
[13] "India" "Iran"  "Iraq"  "Italy" "Japan" "Kenya" "Korea" "Laos"  "Libya" "Mali"  "Malta" "Nepal"
[25] "Niger" "Oman"  "Peru"  "Qatar" "Spain" "Sudan" "Syria" "Togo"  "USA"   "USSR"  "Zaire"
```

all the countries with 15 or more characters in their name
```
> as.vector(Country[grep("^.{15,}$", as.character(Country))])
[1] "Balearic Islands"        "Central African Republic" "Dominican Republic"
[4] "Papua New Guinea"        "Solomon Islands"          "Trinidad & Tobago"
[7] "Tristan da Cunha"
```

搜尋與替換: sub, gsub

- replaces only the first occurrence of a pattern within a character string: `sub(pattern, replacement, x)`
- replace all occurrences: `gsub(pattern, replacement, x)`

```
> text <- c("arm", "leg", "head", "foot", "hand", "hindleg", "elbow")
> text
[1] "arm"     "leg"     "head"    "foot"    "hand"    "hindleg" "elbow"
> gsub("h", "H", text)
[1] "arm"     "leg"     "Head"    "foot"    "Hand"    "Hindleg" "elbow"
> gsub("o", "O", text)
[1] "arm"     "leg"     "head"    "fOOt"    "hand"    "hindleg" "elbOw"
> sub("o", "O", text)
[1] "arm"     "leg"     "head"    "fOot"    "hand"    "hindleg" "elbOw"
> gsub("^.", "O", text)
[1] "Orm"     "Oeg"     "Oead"    "Ooot"    "Oand"    "Oindleg" "Olbow"
```

`replace {base}`: Replace Values in a Vector

`replace(x, list, values)`

```
> x <- c(3, 2, 1, 0, 4, 0)
> replace(x, x==0, 1)
[1] 3 2 1 1 4 1
```

```
> replace(text, text == "leg", "LEG")
[1] "arm"     "LEG"     "head"    "foot"    "hand"    "hindleg" "elbow"
> replace(text, text %in% c("leg", "foot"), "LEG")
[1] "arm"     "LEG"     "head"    "LEG"     "hand"    "hindleg" "elbow"
```

樣式的位置: regexpr

> regexpr(pattern, text)

- match location: if the pattern does not appear within the string, return -1

```
> text <- c("arm", "leg", "head", "foot", "hand", "hindleg", "elbow")
> regexpr("o", text)
[1] -1 -1 -1  2 -1 -1  4
attr(,"match.length")
[1] -1 -1 -1  1 -1 -1  1

#which elements of text contained an "o"
> grep("o", text)
[1] 4 7

#extract the character string
> text[grep("o", text)]
[1] "foot"  "elbow"

#how many "o"s there are in each string
> gregexpr("o", text)
[[1]]
[1] -1
attr(,"match.length")
[1] -1
...
```

```
[[4]]
[1] 2 3
attr(,"match.length")
[1] 1 1
attr(,"index.type")
[1] "chars"
attr(,"useBytes")
[1] TRUE
```

```
#multiple match return 0
> charmatch("m", c("mean","median","mode"))
[1] 0

#unique match return index
> charmatch("med", c("mean","median","mode"))
[1] 2
```

which: Which indices are TRUE?

```
> stock <- c("car", "van")
> requests <- c("truck", "suv", "van", "sports", "car", "waggon", "car")
> requests %in% stock
[1] FALSE FALSE  TRUE FALSE  TRUE FALSE  TRUE
> index <- which(requests %in% stock)
> requests[index]
[1] "van" "car" "car"
```

```
> x <- round(rnorm(10), 2)
> x
 [1] -1.17 -0.05  0.57  0.72 -1.79  0.55  0.03  0.09 -1.81  0.04
> index <- which(x < 0)
> index
[1] 1 2 5 9
> x[index]
[1] -1.17 -0.05 -1.79 -1.81
> x[x < 0]
[1] -1.17 -0.05 -1.79 -1.81
```

`which.max()` #locates first maximum of a numeric vector
`which.min()` #locates first minimum of a numeric vector

課堂練習14

```
> x <- matrix(sample(1:12), ncol=4, nrow=3)
> x
     [,1] [,2] [,3] [,4]
[1,]   12    4    2    8
[2,]    6   10    5    9
[3,]    1    3    7   11
> which(x %% 3 == 0)
[1]  1  2  6 11
> which(x %% 3 == 0, arr.ind = T)
     row col
[1,]   1   1
[2,]   2   1
[3,]   3   2
[4,]   2   4
```

See also:

```
any(..., na.rm = FALSE)
all(..., na.rm = FALSE)
```

```
> x <- c(45, 3, 50, 41, 14, 50, 3)
> which.min(x)
[1] 2
> which.max(x)
[1] 3
> x[which.min(x)]
[1] 3
> x[which.max(x)]
[1] 50
> which(x == max(x))
[1] 3 6
```

```
> match(1:10, 4)
 [1] NA NA NA  1 NA NA NA NA NA NA
> match(1:10, c(4, 2))
 [1] NA  2 NA  1 NA NA NA NA NA NA
> x
[1] 45  3 50 41 14 50  3
> match(x, c(50, 3))
[1] NA  2  1 NA NA  1  2
```

sets {base}: 集合運算

```
> setA <- c("a","b","c", "d", "e")
> setB <- c("d", "e", "f", "g")

> union(setA, setB)
[1] "a" "b" "c" "d" "e" "f" "g"

> intersect(setA, setB)
[1] "d" "e"

> setdiff(setA, setB)
[1] "a" "b" "c"

> setdiff(setB, setA)
[1] "f" "g"

> setA %in% setB
[1] FALSE FALSE FALSE  TRUE  TRUE

> setB %in% setA
[1]  TRUE  TRUE FALSE FALSE

> setA[setA %in% setB] #intersect(setA, setB)
[1] "d" "e"
```

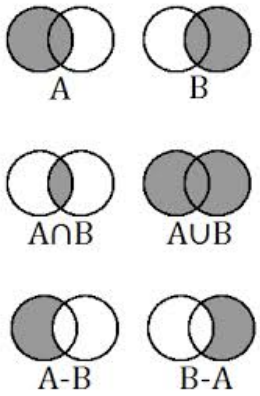
sets {base}: Set Operations

Description: Performs set union, intersection, (asymmetric!) difference, equality and membership on two vectors.

Usage:

```
union(x, y)
intersect(x, y)
setdiff(x, y)
setequal(x, y)
is.element(el, set)
```

`is.element(x, y)` is identical to `x %in% y`.



R程式執行時間

profvis: Interactive Visualizations for Profiling R Code

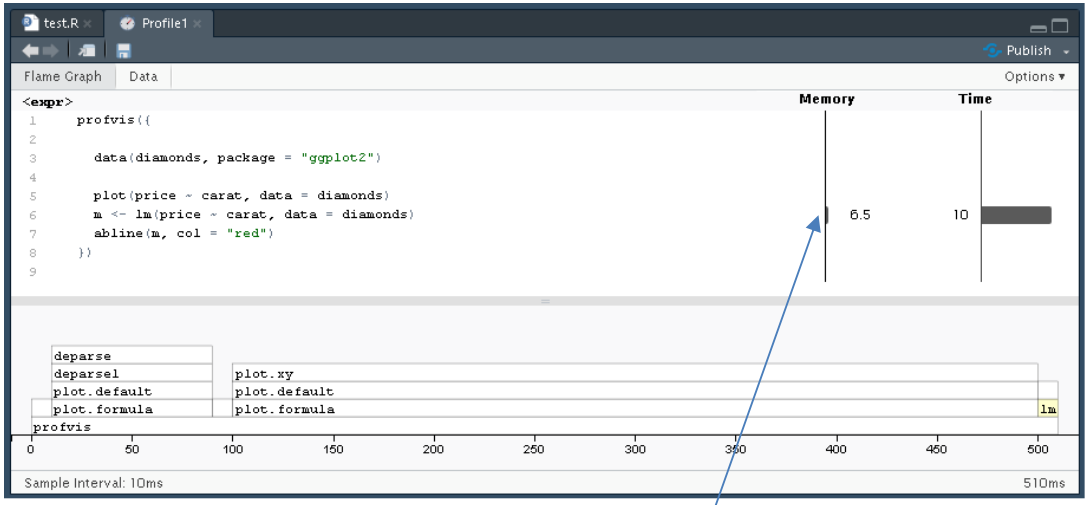
```
myFun <- function(n){
  x <- 0
  for(i in 1:n){
    x <- x + i
  }
  x
}
```

```
> system.time({
+     ans <- myFun(10000)
+ })
   user  system elapsed
   0.04    0.00    0.05
>
```

```
> start.time <- proc.time()
> for(i in 1:50) mad(runif(500))
> proc.time() - start.time
   user  system elapsed
   0.04    0.01    0.05
```

```
> start.time <- Sys.time()
> ans <- myFun(10000)
> end.time <- Sys.time()
> end.time -start.time
Time difference of 0.0940001 secs
```

```
install.packages("profvis")
library(profvis)

profvis({

  data(diamonds, package = "ggplot2")

  plot(price ~ carat, data = diamonds)
  m <- lm(price ~ carat, data = diamonds)
  abline(m, col = "red")
})
```



Memory allocated or deallocated (for negative numbers)

More examples: https://rstudio.github.io/profvis/examples.html

排序: Rank, Sort and Order

```
> city <- read.table("city.txt", header=TRUE, row.names=NULL, sep="\t")
> attach(city)
> names(city)
[1] "location" "price"
>
> rank.price <- rank(price)
> sorted.price <- sort(price)
> ordered.price <- order(price)
```

	A	B
1	location	price
2	Taipei	325
3	New York	201
4	Boston	157
5	Tokyo	162
6	Hong Kong	164
7	Shanghai	95
8	LA	117
9	Vancouver	188
10	Seoul	121
11	Seattle	101

- **order** returns an integer vector containing the permutation that will sort the input into ascending order.
- **order** is useful in sorting dataframes.
- **x[order(x)]** is the same as **sort(x)**

```
> sort(price, decreasing=TRUE)
 [1] 325 201 188 164 162 157 121 117 101  95
> rev(sort(price))
 [1] 325 201 188 164 162 157 121 117 101  95
```

排序: Rank, Sort and Order

```
> city
    location price
1      Taipei   325
2    New York   201
3      Boston   157
4       Tokyo   162
5   Hong Kong   164
6    Shanghai    95
7          La   117
8   Vancouver   188
9       Seoul   121
10    Seattle   101
```

```
> (view1 <- data.frame(location, price, rank.price))
    location price rank.price
1      Taipei   325         10
2    New York   201          9
3      Boston   157          5
4       Tokyo   162          6
5   Hong Kong   164          7
6    Shanghai    95          1
7          LA   117          3
8   Vancouver   188          8
9       Seoul   121          4
10    Seattle   101          2
```

```
> (view2 <- data.frame(sorted.price, ordered.price))
   sorted.price ordered.price
1            95             6
2           101            10
3           117             7
4           121             9
5           157             3
6           162             4
7           164             5
8           188             8
9           201             2
10          325             1
```

```
> (view3 <- data.frame(location[ordered.price],
price[ordered.price]))
   location.ordered.price. price.ordered.price.
1                 Shanghai                    95
2                  Seattle                   101
3                       LA                   117
4                    Seoul                   121
5                   Boston                   157
6                    Tokyo                   162
7                Hong Kong                   164
8                Vancouver                   188
9                 New York                   201
10                  Taipei                   325
```

See: multiple sorting, text sorting
http://rprogramming.net/r-order-to-sort-data/

x[order(x)] is the same as sort(x

資料處理、表格相關

Sampling without replacement

```
> y <- 1:20
> sample(y)
> sample(y)
> sample(y, 5)
> sample(y, 5)
> sample(y, 5, replace=T)
```

Substrings

```
> substr("this is a test", start=1, stop=4)
> substr(rep("abcdef",4),1:4,4:5)

> x <- c("asfef", "qwerty", "yuiop[", "b", "stuff.blah.yech")
> substr(x, 2, 5)
> substring(x, 2, 4:6)
> substring(x, 2) <- c("..", "+++")
> x
```

See also:

- B01-1-hmwu_R-DataManipulation.pdf
- stack {utils}, reshape {stats}, melt{reshape}, cast{reshape}, merge {base}, sample {base}, subset {base}
- xtabs {stats}, table {base}, tabulate {base}, ftable {stats}, xtable{xtable}

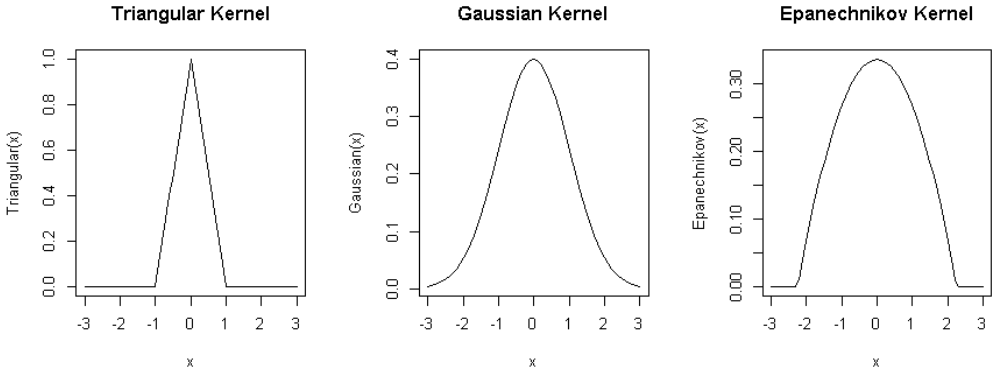
Arguments 為函數

以下三個核函數 (kernel function) 是在進行核密度函數估計中常用的函數。若 `u <- seq(-3, 3, 0.1)`, 請畫出kernel 圖形。

Kernel	Function
Triangular	$K(u) = (1 - \|u\|)I(\|u\| \leq 1)$
Gaussian	$K(u) = \frac{1}{\sqrt{2\pi}} \exp(-\frac{1}{2}u^2)$
Epanechnikov	$K(u) = \frac{3}{4\sqrt{5}}(1 - \frac{u^2}{5})I(\|u\| \leq \sqrt{5})$

其中 $I(|u| \leq a) = \begin{cases} 1, & \text{if } |u| \leq a \\ 0, & \text{if } |u| > a \end{cases}$

```
Triangular <- function(u){
          s <- ifelse(abs(u) <= 1, 1, 0)
          ans <- (1-abs(u))*s
          ans
}
Gaussian <- function(u){
          ans <- exp((-1/2)*(u^2))/sqrt(2*pi)
          ans
}
Epanechnikov <- function(u){
          s <- ifelse(abs(u) <= sqrt(5), 1, 0)
          ans <- 3*(1-((u^2)/5))/(4*sqrt(5))*s
          ans
```




```
> par(mfrow=c(1,3))
> x <- seq(-3, 3, 0.1)
> plot(x, Triangular(x), main="Triangular Kernel", type="l")
> plot(x, Gaussian(x), main="Gaussian Kernel", type="l")
> plot(x, Epanechnikov(x), main="Epanechnikov Kernel", type="l")
```

課堂練習: 請改用 `apply(as.matrix(x)...`

Arguments 為函數

Let $x_1, x_2, \cdots, x_n$ be an iid sample drawn from some distribution with an unknown density f. We are interested in estimating the shape of this function f. Its kernel density estimator is

$$\hat{f}_h(x) = \frac{1}{nh} \sum_{i=1}^{n} K(\frac{x - x_i}{h})$$

with kernel K and bandwidth h。

若觀察資料 $x_1, x_2, \cdots, x_n$爲 `xi <- iris[,1]`, 試寫一 R 函式, 計算$\hat{f}_h(x)$其在$x = 7$, $h = 0.2736$之下, 使用上述三種 kernel 之值。

```
fh <- function(xi, x, h, kernel, n=150){
          ans <- sum(kernel((x-xi)/h))/(n*h)
          ans
}
```

```
> xi <- iris[, 1]
> fh(xi, x = 7, h = 0.2736, Triangular)
[1] 0.1409978
> fh(xi, x = 7, h = 0.2736, Gaussian)
[1] 0.179705
> fh(xi, x = 7, h = 0.2736, Epanechnikov)
[1] 0.1777105
```

印出Arguments為函數的名稱

```
binomial <- function(k, n, p){
  factorial(n)/(factorial(k) * factorial(n - k)) * (p^k) * ((1-p)^(n-k))
}

compute_mu_sigma <- function(pmf, parameter){

  mu <- 0
  sigma2 <- 1

  pmf.name <- deparse(substitute(pmf))
  cat("Input is", pmf.name, "distribution.\n")
  if(pmf.name ==  "binomial"){
    # 讀取參數
    k <- parameter[[1]]
    n <- parameter[[2]]
    p <- parameter[[3]]
    mu <- sum(k * pmf(k, n, p))
    sigma2 <- sum((k - mu)^2 * pmf(k, n, p))
  }
  cat("mu: ", mu, "\n")
  cat("sigma2: ", sigma2, "\n")
}
```

Notation	$B(n, p)$
Parameters	$n \in \mathbb{N}_0$ — number of trials $p \in [0,1]$ — success probability in each trial
Support	$k \in \{0, \ldots, n\}$ — number of successes
pmf	$\binom{n}{k} p^k (1-p)^{n-k}$
CDF	$I_{1-p}(n-k, 1+k)$
Mean	np
Median	$\lfloor np \rfloor$ or $\lceil np \rceil$
Mode	$\lfloor (n+1)p \rfloor$ or $\lceil (n+1)p \rceil - 1$
Variance	$np(1-p)$

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

$$E(X) = \mu = \sum_{x \in D} x \cdot f(x)$$

$$\sigma^2 = Var(X) = \sum_{x \in D} (x - \mu)^2 f(x)$$

```
> compute_mu_sigma(pmf = binomial, parameter = c(4, 10, 0.5))
Input is binomial distribution.
mu:  0.8203125
sigma2:  2.073424
```

do.call {base}: Execute a Function Call

```
binomial <- function(k, n, p){
  factorial(n)/(factorial(k) * factorial(n - k)) * (p^k) * ((1-p)^(n-k))
}
poisson <- function(k, lambda){
  exp(-lambda) * (lambda^k)/(factorial(k))
}
geometric <- function(k, p){
  (1 - p)^k * p
}
compute_mu_sigma <- function(pmf, parameter){
  pmf.name <- deparse(substitute(pmf))
  mu <- sum(parameter$k * (do.call("pmf", parameter)))
  sigma2 <- sum((parameter$k - mu)^2 * do.call("pmf", parameter))
  cat("distribution:  ", pmf.name, "\n")
  cat("mu: ", mu, "\t sigma2:", sigma2, "\n" )
}
```

Description: do.call constructs and executes a function call from a name or a function and a list of arguments to be passed to it.

Usage: `do.call(what, args, quote = FALSE, envir = parent.frame())`

```
> my.par <- list(k = c(0:10), n = 10, p = 0.6)
> compute.mu.sigma(pmf = binomial, parameter = my.par)
distribution:   binomial
mu:  6   sigma2: 2.4
> my.par <- list(k = c(0:100), lambda = 4)
> compute.mu.sigma(pmf = poisson, parameter = my.par)
distribution:   poisson
mu:  4   sigma2: 4
> my.par <- list(k = c(0:10000), p = 0.4)
> compute_mu_sigma(pmf = geometric, parameter = my.par)
distribution:   geometric
mu:  1.5         sigma2: 3.75
```

物件屬性強制轉換 (Coercing)

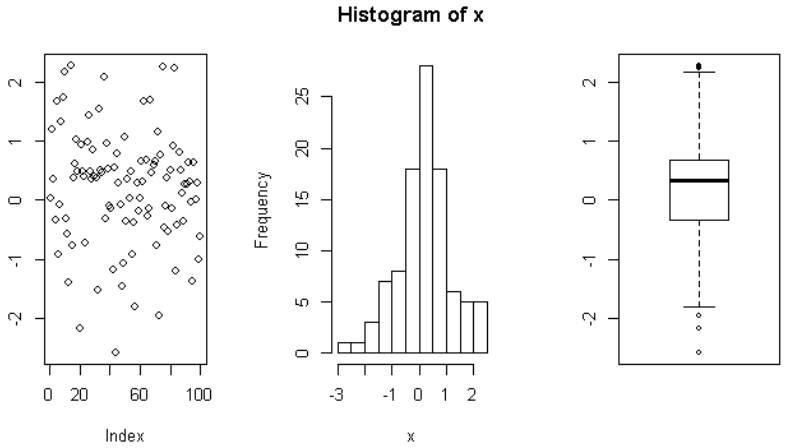
Table 2.4. Functions for testing (is) the attributes of different categories of object (arrays, lists, etc.) and for coercing (as) the attributes of an object into a specified form. Neither operation changes the attributes of the object.

Type	Testing	Coercing
Array	is.array	as.array
Character	is.character	as.character
Complex	is.complex	as.complex
Dataframe	is.data.frame	as.data.frame
Double	is.double	as.double
Factor	is.factor	as.factor
List	is.list	as.list
Logical	is.logical	as.logical
Matrix	is.matrix	as.matrix
Numeric	is.numeric	as.numeric
Raw	is.raw	as.raw
Time series (ts)	is.ts	as.ts
Vector	is.vector	as.vector

```
as.numeric(factor(c("a", "b", "c")))
as.numeric(c("a", "b", "c")) #don't work
```

字串轉成變數名稱或指令

```
> (x <- sample(1:42, 6))
[1]  3  1 29 16 36 21
> (y <- letters)
> get("x")
[1]  3  1 29 16 36 21
> get("y")[1:5]
[1] "a" "b" "c" "d" "e"
>
> for(i in 1:5){
+     x.name <- paste("x", i, sep=".")
+     assign(x.name, 1:i)
+     cat(x.name, ": \t")
+     cat(get(x.name), "\n")
+ }
x.1 :   1
x.2 :   1 2
x.3 :   1 2 3
x.4 :   1 2 3 4
x.5 :   1 2 3 4 5
```




```
> a <- 100
> (my.math <- c("3 + 4", "a / 5"))
[1] "3 + 4" "a / 5"
> eval(my.math)
[1] "3 + 4" "a / 5"
> eval(parse(text = my.math[1]))
[1] 7
>
> plot.type <- c("plot", "hist", "boxplot")
> x <- rnorm(100)
> my.plot <- paste(plot.type, "(x)", sep = "")
> eval(parse(text = my.plot))
```

eval() evaluates an expression, but "5+5" is a string, not an expression. So, use parse() with text= to translate the string to an expression

查看指令程式碼

```
> library(e1071)
> fclustIndex
function (y, x, index = "all")
{
    clres <- y
    gath.geva <- function(clres, x) {
        xrows <- dim(clres$me)[1]
        xcols <- dim(clres$ce)[2]
        ncenters <- dim(clres$centers)[1]
        scatter <- array(0, c(xcols, xcols, ncenters))
...
```

```
> plot
function (x, y, ...)
UseMethod("plot")
<bytecode: 0x000000000fdaaad8>
<environment: namespace:graphics>
> methods(plot)
 [1] plot.acf*           plot.agnes* ... plot.lm     ... plot.table ...

   Non-visible functions are asterisked
> plot.lm
function (x, which = c(1L:3L, 5L), caption = list("Residuals vs Fitted",
    "Normal Q-Q", "Scale-Location", "Cook's distance", "Residuals vs Leverage",
    expression("Cook's dist vs Leverage  " * h[ii]/(1 - h[ii]))),
```

查看指令程式碼

```
> plot.table
錯誤: 找不到物件 'plot.table'
> ?plot.table                          #plot.table {graphics}
> graphics:::plot.table
function (x, type = "h", ylim = c(0, max(x)), lwd = 2, xlab = NULL,
    ylab = NULL, frame.plot = is.num, ...)
{
    xnam <- deparse(substitute(x))
    rnk <- length(dim(x))
    if (rnk == 0L)
        stop("invalid table 'x'")
    if (rnk == 1L) {
    . . .
```

Accessing exported and internal variables, i.e. R objects in a namespace.

pkg::name

pkg:::name

```
> anova
> methods(anova)
> stats:::anova.nls
> stats:::anova.loess
```

```
> princomp
> methods(princomp)
> getAnywhere('princomp.default') # or
> stats:::princomp.default
```

```
> svm
> ?svm
> methods(svm)
> e1071:::svm.default
. . .
   cret <- .C("svmtrain", as.double(if (sparse) x@ra else t(x)),
        as.integer(nr), as.integer(nco
. . .
```

Package source: e1071_1.6-3.tar.gz

Windows binaries: e1071_1.6-3.zip

```
e1071_1.6-3\e1071\src\
cmeans.c, cshell.c, floyd.c, Rsvm.c, svm.cpp, svm.h, ...
e1071_1.6-3\e1071\R\
bclust.R, bincombinations.R, cmeans.R, fclustIndex.R, ...
```

Reference Card

R Reference Card 2.0

Public domain, v2.0 2012-12-24.
V 2 by Matt Baggott, matt@baggott.net
V 1 by Tom Short, t.short@ieee.org
Material from *R for Beginners* by permission of Emmanuel Paradis.

Getting help and info

help(topic) documentation on topic
?topic same as above; special chars need quotes: for example **?'&&'**
help.search("topic") search the help system; same

Operators

<-	Left assignment, binary
->	Right assignment, binary
=	Left assignment, but not recommended
<<-	Left assignment in outer lexical scope; not for beginners
$	List subset, binary
-	Minus, can be unary or binary
+	Plus, can be unary or binary
~	Tilde, used for model formulae
:	Sequence, binary (in model formulae: interaction)
::	Refer to function in a package, i.e,

R Reference Card
(Version 2)

R Reference Card for Data Mining (2015)
http://www.rdatamining.com/docs/r-reference-card-for-data-mining

Contributed Documentation
http://cran.r-project.org/other-docs.html

R Functions List (+ Examples) | All Basic Commands of the R Programming Language
https://statisticsglobe.com/r-functions-list/

R Reference Card for Data Mining

Yanchang Zhao, RDataMining.com, January 8, 2015

- See the latest version at `http://www.RDataMining.com`
- The package names are in parentheses.
- Recommended packages and functions are shown in bold.
- Click a package in this PDF file to find it on CRAN.

Association Rules and Sequential Patterns

Functions

`apriori()` mine associations with APRIORI algorithm – a level-wise, breadth-first algorithm which counts transactions to find frequent itemsets (*arules*)
`eclat()` mine frequent itemsets with the Eclat algorithm, which employs equivalence classes, depth-first search and set intersection instead of counting (*arules*)
`cspade()` mine frequent sequential patterns with the cSPADE algorithm (*arulesSequences*)
`seqefsub()` search for frequent subsequences (*TraMineR*)

Packages

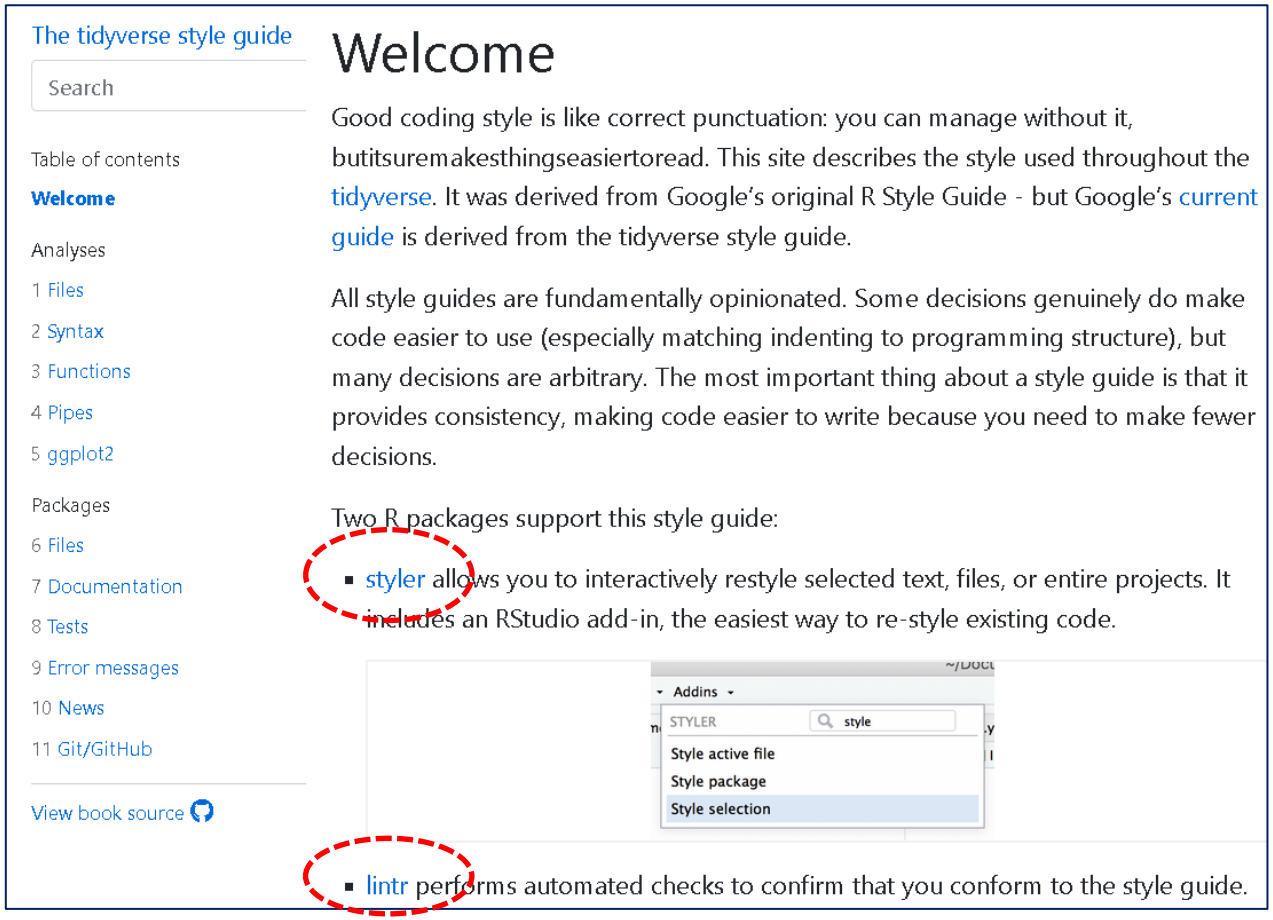
arules mine frequent itemsets, maximal frequent itemsets, closed frequent itemsets and association rules. It includes two algorithms, Apriori and Eclat.
arulesViz visualizing association rules
arulesSequences add-on for *arules* to handle and mine frequent sequences
TraMineR mining, describing and visualizing sequences of states or events

Classification & Prediction

R程式風格: 邁向專業的R程式設計

Tidyverse Style Guide by Hadley Wickham
https://style.tidyverse.org

Google's R Style Guide
https://google.github.io/styleguide/Rguide.xml



The tidyverse style guide

Search

Table of contents

Welcome

Analyses

1 Files
2 Syntax
3 Functions
4 Pipes
5 ggplot2

Packages

6 Files
7 Documentation
8 Tests
9 Error messages
10 News
11 Git/GitHub

View book source

Welcome

Good coding style is like correct punctuation: you can manage without it, butitsuremakesthingseasiertoread. This site describes the style used throughout the tidyverse. It was derived from Google's original R Style Guide - but Google's current guide is derived from the tidyverse style guide.

All style guides are fundamentally opinionated. Some decisions genuinely do make code easier to use (especially matching indenting to programming structure), but many decisions are arbitrary. The most important thing about a style guide is that it provides consistency, making code easier to write because you need to make fewer decisions.

Two R packages support this style guide:

- styler allows you to interactively restyle selected text, files, or entire projects. It includes an RStudio add-in, the easiest way to re-style existing code.

- lintr performs automated checks to confirm that you conform to the style guide.

Google Python Style Guide
https://google.github.io/styleguide/pyguide.html: https://tw-google-styleguide.readthedocs.io/en/latest/google-python-styleguide/
Google Java Style Guide: https://google.github.io/styleguide/javaguide.html

程式風格，styler和lintr套件

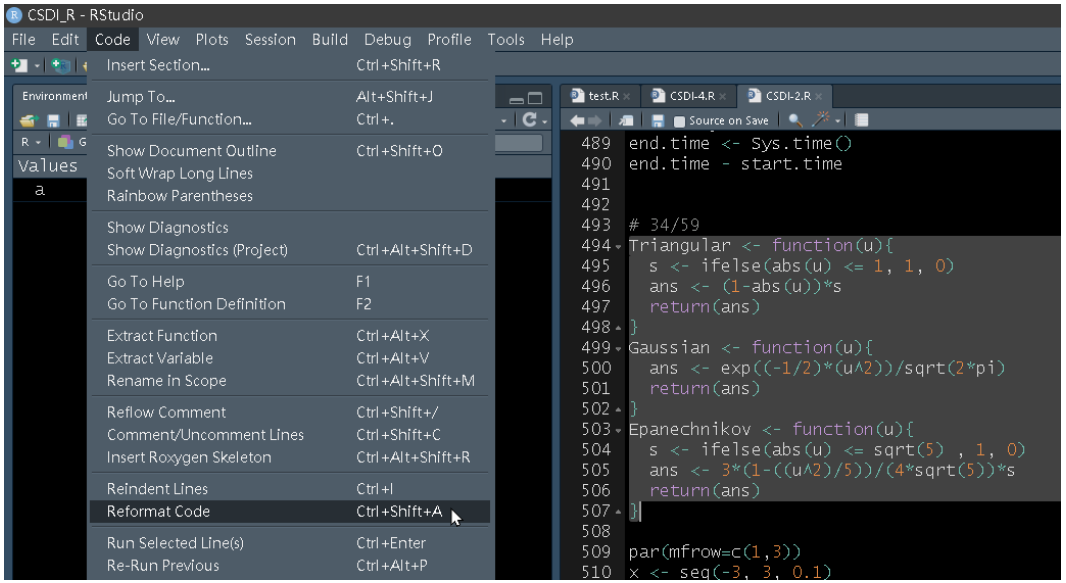
❑ Paul E. Johnson, R Style. An Rchaeological Commentary
http://cran.r-project.org/web/packages/rockchalk/vignettes/Rstyle.pdf

```
nor=matrix(rnorm(Nsim*p),nrow=p)
risk=matrix(0,ncol=150,nrow=10)
a=seq(1,2*(p-2),le=10)
the=sqrt(seq(0,4*p,le=150)/p)
for (j in 1:150){
nornor=apply((nor+rep(the[j],p))^2,2,sum)
for (i in 1:10){
for (t in 1:Nsim)
risk[i,j]=risk[i,j]+sum((rep(the[j],p)-
max(1-a[i]/nornor[t],0)*(nor[,t]+rep(the[j],p)))^2)
}}
risk=risk/Nsim
```

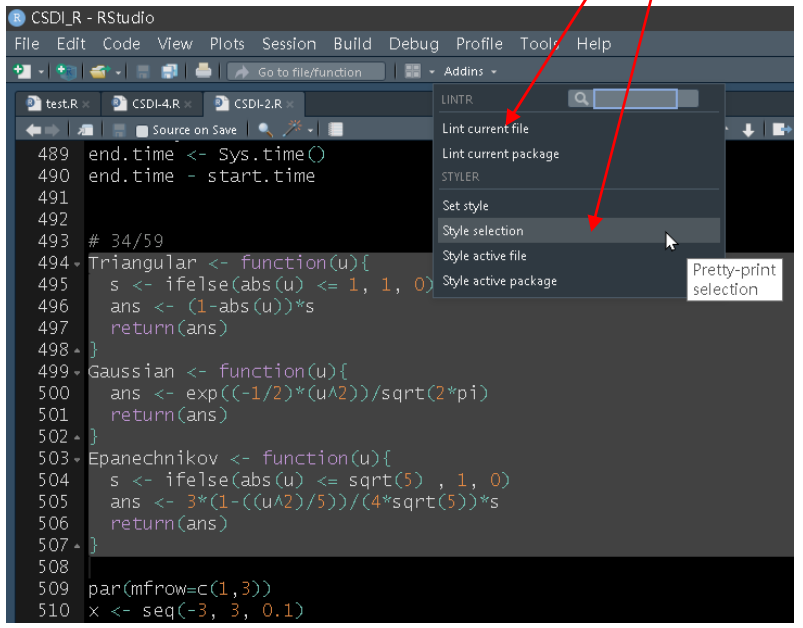
❑ R Coding Conventions (RCC)
http://www.aroma-project.org/developers/RCC

```
nor <- matrix(rnorm(Nsim*p), nrow=p)
risk <- matrix(0, ncol=150, nrow=10)
a <- seq(1, 2*(p-2), le=10)
the <- sqrt(seq(0, 4*p, le=150)/p)
for(j in 1:150){
    nornor <- apply((nor + rep(the[j],p))^2, 2, sum)
    for(i in 1:10){
        for(t in 1:Nsim){
            s <- max(1-a[i]/nornor[t], 0)
            t <- nor[,t] + rep(the[j], p)
            u <- sum((rep(the[j], p) - s*t)^2)
            risk[i,j] <- risk[i,j] + u
        }
    }
}
risk <- risk/Nsim
```

RStudio: Reformat Code



Addins: styler和lintr套件



請不要寫這種「程式」！

```
d11 <- ((x1-x2)^2+(y1-y2)^2)^(1/2)
d11

d12 <- ((x1-x3)^2+(y1-y3)^2)^(1/2)
d12

d13 <- ((x1-x4)^2+(y1-y4)^2)^(1/2)
d13

d14 <- ((x1-x5)^2+(y1-y5)^2)^(1/2)
d14

d21 <- ((x2-x3)^2+(y2-y3)^2)^(1/2)
d21

d22 <- ((x2-x4)^2+(y2-y4
d22

d23 <- ((x2-x5)^2+(y2-y5
d23

d31 <- ((x3-x4)^2+(y3-y4
d31

d32 <- ((x3-x5)^2+(y3-y5
d32

d41 <- ((x4-x5)^2+(y4-y5
d41
```

```
> if(
sqrt((m8[1,1]-m8[2,1])^2+(m8[1,2]-m8[2,2])^2)+
sqrt((m8[2,1]-m8[3,1])^2+(m8[2,2]-m8[3,2])^2)>
sqrt((m8[3,1]-m8[1,1])^2+(m8[3,2]-m8[1,2])^2)
&
sqrt((m8[1,1]-m8[2,1])^2+(m8[1,2]-m8[2,2])^2)+
sqrt((m8[3,1]-m8[1,1])^2+(m8[3,2]-m8[1,2])^2)>
sqrt((m8[2,1]-m8[3,1])^2+(m8[2,2]-m8[3,2])^2)
&
sqrt((m8[2,1]-m8[3,1])^2+(m8[2,2]-m8[3,2])^2)+
sqrt((m8[3,1]-m8[1,1])^2+(m8[3,2]-m8[1,2])^2)>
sqrt((m8[1,1]-m8[2,1])^2+(m8[1,2]-m8[2,2])^2)
){
```

```
(x == my.data[1,1] ) |(x == my.data[1,2] )|(x == my.data[1,3] )|(x ==
my.data[1,4] )|(x == my.data[1,5] )|(x == my.data[1,6] )
第一筆 <-
c(my.data[1,1],my.data[1,2],my.data[1,3],my.data[1,4],my.data[1,5],my.data[1,6])

(x == my.data[2,1] ) |(x == my.data[2,2] )|(x == my.data[2,3] )|(x ==
my.data[2,4] )|(x == my.data[2,5] )|(x == my.data[2,6] )
第二筆 <-
c(my.data[2,1],my.data[2,2],my.data[2,3],my.data[2,4],my.data[2,5],my.data[2,6])


(x == my.data[3,1] ) |(x == my.data[3,2] )|(x == my.data[3,3] )|(x ==
my.data[3,4] )|(x == my.data[3,5] )|(x == my.data[3,6] )
第三筆 <-
c(my.data[3,1],my.data[3,2],my.data[3,3],my.data[3,4],my.data[3,5],my.data[3,6])



(x == my.data[4,1] ) |(x == my.data[4,2] )|(x == my.data[4,3] )|(x ==
my.data[4,4] )|(x == my.data[4,5] )|(x == my.data[4,6] )
第四筆 <-
c(my.data[4,1],my.data[4,2],my.data[4,3],my.data[4,4],my.data[4,5],my.data[4,6])
```

Tidyverse Style Guide: Files
Names, Organisation, Internal structure

https://style.tidyverse.org

The tidyverse style guide

Search

Table of contents

Welcome

Analyses

1 Files

2 Syntax

3 Functions

4 Pipes

5 ggplot2

Packages

6 Files

7 Documentation

8 Tests

9 Error messages

10 News

11 Git/GitHub

- **File names**: be meaningful.

```
# Good: fit_models.R, utility_functions.R
# Bad: fit models.R, foo.r, stuff.r
```

- **File names with orders**: all lower case.

```
00_download.R
01_explore.R
02a_plot.R
02b_summarize.R
...
09_model.R
10_visualize.R
```

If your script uses add-on packages, load them `library()` all at once at the very beginning of the file.

- **Internal structure of a R file:** Use commented lines of `-` and `=`

```
# Load data ---------------------------

# Plot data ===========================
```

Line Length: the maximum line length is 80 characters.

Tidyverse Style Guide: Syntax
Object names

- **Object names**: Variable and function names should use only lowercase letters, numbers, and _

```
# Good: day_one; day_1
# Bad: DayOne; dayone
```

✓ Note: Base R uses dots in function names (`contrib.url()`) and class names (`data.frame`), but it's better to reserve dots exclusively for the S3 object system.

- **Object names**: Variable names should be nouns and function names should be verbs

```
# Good: day_one
# Bad: first_day_of_the_month; djm1
```

- **Object names**: Avoid re-using names of common functions and variables:

```
# Bad:
T <- FALSE
c <- 10
mean <- function(x) sum(x)
```

Tidyverse Style Guide: Syntax
Spacing: Commas, Parentheses

- **Commas**: Always put a space after a comma

```
# Good: x[, 1]
# Bad: x[,1]; x[ ,1]; x[ , 1]
```

- **Parentheses**: Do not put spaces inside or outside parentheses for regular function calls.

```
# Good: mean(x, na.rm = TRUE)
# Bad: mean (x, na.rm = TRUE); mean( x, na.rm = TRUE )
```

- **Parentheses**: Place a space before and after `()` when used with if, for, or while.

```
# Good:
if (debug) {
  show(x)
}
# Bad:
if(debug){
  show(x)
}
```

- **Parentheses**: Place a space after `()` used for function arguments:

```
# Good: function(x) {}
# Bad: function (x) {}; function(x){}
```

Tidyverse Style Guide: Syntax
Spacing: Infix operators

- **Infix operators**: (`==, +, -, *, /, <-`, etc.) should always be surrounded by spaces

```
# Good:
height <- (feet * 12) + inches
mean(x, na.rm = TRUE)

# Bad:
height<-feet*12+inches
mean(x, na.rm=TRUE)
```

- **Infix operators (exceptions)**: high precedence: `::, :::, $, @, [, [[, ^`, unary `-`, unary `+`, and `:`.

```
# Good:
sqrt(x^2 + y^2)
df$z
x <- 1:10

# Bad:
sqrt(x ^ 2 + y ^ 2)
df $ z
x <- 1 : 10
```

- **Infix operators (exceptions)**: bang tidy evaluation

```
# Good: call(!!xyz)
# Bad: call(!! xyz); call( !! xyz); call(! !xyz)
```

- **Infix operators (exceptions)**: The help operator

```
# Good: ?mean
# Bad: ? mean
```

- **Infix operators (exceptions)**: Single-sided formulas when the right-hand side is a single identifier:

```
# Good: ~foo
# Bad: ~ foo
```

✓ Note: single-sided formulas with a complex right-hand side do need a space:

```
# Good: ~ .x + .y
# Bad: ~.x + .y
```

Tidyverse Style Guide: Syntax
Spacing: Embracing, Extra spaces

- **The embracing operator**, `{{ }}`: always have inner spaces to help emphasise its special behaviour:

```
# Good: group_by({{ by }})
# Bad: group_by({{by}})
```

- **Extra spaces**: Adding extra spaces is ok if it improves alignment of `=` or `<-`.

```
# Good
list(
  total = a + b + c,
  mean  = (a + b + c) / n
)

# Also fine
list(
  total = a + b + c,
  mean = (a + b + c) / n
)
```

Tidyverse Style Guide: Syntax
Function calls

- **Named arguments:**
  - A function's arguments typically fall into two broad categories: one supplies the data to compute on; the other controls the details of computation.
  - When you call a function, you typically omit the names of data arguments, because they are used so commonly. If you override the default value of an argument, use the full name:

```
# Good:
mean(1:10, na.rm = TRUE)

# Bad:
mean(x = 1:10, , FALSE)
mean(, TRUE, x = c(1:10, NA))
```

- **Assignment**: Avoid assignment in function calls:

```
# Good:
x <- complicated_function()
if (nzchar(x) < 1) {
  # do something
}

# Bad:
if (nzchar(x <- complicated_function()) < 1) {
  # do something
}
```

Tidyverse Style Guide: Syntax
Control flow: Code blocks

❑ Code blocks:

- `{` should be the last character on the line.
- The contents should be indented by two spaces.
- `}` should be the first character on the line.

```
# Good:
if (y < 0 && debug) {
  message("y is negative")
}

if (y == 0) {
  if (x > 0) {
    log(x)
  } else {
    message("x is negative or zero")
  }
} else {
  y^x
}

test_that("call1 returns an ordered factor", {
  expect_s3_class(call1(x, y), c("factor", "ordered"))
})

tryCatch(
  {
    x <- scan()
    cat("Total: ", sum(x), "\n", sep = "")
  },
  interrupt = function(e) {
    message("Aborted by user")
  }
)
```

```
# Bad:
if (y < 0 && debug) {
message("Y is negative")
}

if (y == 0)
{
    if (x > 0) {
      log(x)
    } else {
  message("x is negative or zero")
    }
} else { y ^ x }
```

Tidyverse Style Guide: Syntax
Control flow: `if` , `switch`, Inline statements

❑ If statements

- If used, `else` should be on the same line as `}`.
- `&` and `|` should never be used inside of an if clause because they can return vectors. Always use `&&` and `||` instead.
- If you want to rewrite a simple but lengthy `if` block , just write it all on one line

```
# Good:
message <- if (x > 10) "big" else "small"

# Bad:
if (x > 10) {
  message <- "big"
} else {
  message <- "small"
}
```

❑ Implicit type coercion: Avoid implicit type coercion (e.g. from numeric to logical) in if statements:

```
# Good:
if (length(x) > 0) {
  # do something
}
```

```
# Bad:
if (length(x)) {
  # do something
}
```

❑ Inline statements

```
# Good:
y <- 10
if (y < 0) {
  stop("Y is nega. ")
}

find_abs <- function(x) {
  if (x > 0) {
    return(x)
  }
  x * -1
}
```

```
# Bad:
if (y < 0) stop("Y is nega.")

if (y < 0)
  stop("Y is negative")

find_abs <- function(x) {
  if (x > 0) return(x)
  x * -1
}
```

❑ Switch statements

```
# Good:
switch(x,
  a = ,
  b = 1,
  c = 2,
  stop("Unknown `x`", call. = FALSE)
)

# Bad:
switch(x, a = , b = 1, c = 2)
switch(x, a =, b = 1, c = 2)
switch(y, 1, 2, 3)
```

Tidyverse Style Guide: Syntax

Long lines, Semicolons, Assignment, Data, Comments

- **Long lines**: Strive to limit your code to 80 characters per line

```
# Good:
do_something_very_complicated(
  something = "that",
  requires = many,
  arguments = "some of which may be long"
)

# Bad:
do_something_very_complicated("that", requires, many, arguments,
                              "some of which may be long"
                              )
```

- **Semicolons**: Don't put ; at the end of a line, and don't use ; to put multiple commands on one line.

- **Assignment**: Use `<-`, not `=`, for assignment.

```
# Good: x <- 5
# Bad: x = 5
```

- **Character vectors:** Use ", not ', for quoting text.

```
# Good:
"Text"
'Text with "quotes"'
'<a href="http://style.tidyverse.org">A link</a>'

# Bad:
'Text'
'Text with "double" and \'single\' quotes'
```

- **Logical vectors:** Prefer **TRUE** and **FALSE** over **T** and **F.**
- **Comments:** Each line of a comment should begin with the comment symbol and a single space: #

Tidyverse Style Guide: Functions
Naming

- **Naming**: use verbs for function names:

```
# Good:
add_row()
permute()
```

```
# Bad:
row_adder()
permutation()
```

- **Long lines**: Function-indent (縮排，Prefer):

```
# Good:
long_function_name <- function(a = "a long argument",
                               b = "another argument",
                               c = "another long argument") {
  # As usual code is indented by two spaces.
}
# Bad:
long_function_name <- function(a = "a long argument",
  b = "another argument",
  c = "another long argument") {
  # Here it's hard to spot where the definition ends and the
  # code begins, and to see all three function arguments
}
```

- **Long lines**: Double-indent:

```
long_function_name <- function(
    a = "a long argument",
    b = "another argument",
    c = "another long argument") {
  # As usual code is indented by two spaces.
}
```

Tidyverse Style Guide: Functions
return()

- Only use `return()` for early returns. Otherwise, rely on R to return the result of the last evaluated expression.

```
# Good:
find_abs <- function(x) {
  if (x > 0) {
    return(x)
  }
  x * -1
}
add_two <- function(x, y) {
  x + y
}

# Bad:
add_two <- function(x, y) {
  return(x + y)
}
```

- **Return** statements should always be on their own line because they have important effects on the control flow

```
# Good:
find_abs <- function(x) {
  if (x > 0) {
    return(x)
  }
  x * -1
}

# Bad:
find_abs <- function(x) {
  if (x > 0) return(x)
  x * -1
}
```

Tidyverse Style Guide: Functions Comments

- In code, use comments to explain the "why" not the "what" or "how". Each line of a comment should begin with the comment symbol and a single space: #.

```
# Good:
# Objects like data frames are treated as leaves
x <- map_if(x, is_bare_list, recurse)

# Bad:
# Recurse only with bare lists
x <- map_if(x, is_bare_list, recurse)
```

- **Comments** should be in sentence case, and only end with a full stop if they contain at least two sentences:

```
# Good:
# Objects like data frames are treated as leaves
x <- map_if(x, is_bare_list, recurse)

# Do not use `is.list()`. Objects like data frames must be treated
# as leaves.
x <- map_if(x, is_bare_list, recurse)

# Bad:
# objects like data frames are treated as leaves
x <- map_if(x, is_bare_list, recurse)

# Objects like data frames are treated as leaves.
x <- map_if(x, is_bare_list, recurse)
```

Tidyverse Style Guide: Pipes %>%

- Use %>% to emphasise a sequence of actions, rather than the object that the actions are being performed on.
- **Whitespace**: %>% should always have a space before it, and should usually be followed by a new line. After the first step, each line should be indented by two spaces.

```
# Good:
iris %>%
  group_by(Species) %>%
  summarize_if(is.numeric, mean) %>%
  ungroup() %>%
  gather(measure, value, -Species) %>%
  arrange(value)

# Bad:
iris %>% group_by(Species) %>% summarize_all(mean) %>%
ungroup %>% gather(measure, value, -Species) %>%
arrange(value)
```

- **Long lines**: put each argument on its own line and indent:

```
iris %>%
  group_by(Species) %>%
  summarise(
    Sepal.Length = mean(Sepal.Length),
    Sepal.Width = mean(Sepal.Width),
    Species = n_distinct(Species)
  )
```

- **No arguments**:

```
# Good:
x %>%
  unique() %>%
  sort()

# Bad:
x %>%
  unique %>%
  sort
```

- **Assignment**: Variable name and assignment on separate lines

```
iris_long <-
  iris %>%
  gather(measure, value, -Species) %>%
  arrange(-value)
```

Tidyverse Style Guide: ggplot2

- Styling suggestions for **+** used to separate ggplot2 layers are very similar to those for **%>%** in pipelines.

- **Whitespace**: **+** should always have a space before it, and should be followed by a new line. This is true even if your plot has only two layers. After the first step, each line should be indented by two spaces.

```
# Good:
iris %>%
  filter(Species == "setosa") %>%
  ggplot(aes(x = Sepal.Width, y = Sepal.Length)) +
  geom_point()

# Bad:
iris %>%
  filter(Species == "setosa") %>%
  ggplot(aes(x = Sepal.Width, y = Sepal.Length)) +
    geom_point()

# Bad:
iris %>%
  filter(Species == "setosa") %>%
  ggplot(aes(x = Sepal.Width, y = Sepal.Length)) + geom_point()
```

Tidyverse Style Guide: ggplot2

- **Long lines**:

```
# Good:
ggplot(aes(x = Sepal.Width, y = Sepal.Length, color = Species)) +
  geom_point() +
  labs(
    x = "Sepal width, in cm",
    y = "Sepal length, in cm",
    title = "Sepal length vs. width of irises"
  )

# Bad:
ggplot(aes(x = Sepal.Width, y = Sepal.Length, color = Species)) +
  geom_point() +
  labs(x = "Sepal width, in cm", y = "Sepal length, in cm", title = "Sepal length vs.
width of irises")
```

- Do the **data manipulation** in a pipeline before starting plotting.

```
# Good:
iris %>%
  filter(Species == "setosa") %>%
  ggplot(aes(x = Sepal.Width, y = Sepal.Length)) +
  geom_point()

# Bad:
ggplot(filter(iris, Species == "setosa"), aes(x = Sepal.Width, y = Sepal.Length)) +
  geom_point()
```

Tidyverse Style Guide: Documentation https://style.tidyverse.org/documentation.html
Documentation of code is essential, even if the only person using your code is future-you. Use `roxygen2` with `markdown` support enabled to keep your documentation close to the code.

勉勵的話

全文摘錄自: http://www.csie.stu.edu.tw/資料下載/課程資料/計算與邏輯思考/計算與邏輯思考ch2.doc

學資訊相關科系的同學或多或少都要修習一些程式設計的課程，對許多人來說，學習程式設計是一件令人苦惱的事，枯燥的指令與失敗的挫折似乎比趣味與成就感的機會要多的多，而且事實上這些人之中，將來以設計程式為主要工作的比率也並不高。

那麼是否一定要花如此多的時間與精力來學呢？

事實上，程式除了是指揮電腦工作的工具之外，學習程式至少還有三個好處：

1. 學習程式是了解計算機運作原理的最佳途徑。
2. 培養邏輯思考的能力。
3. 構思一個有條理的程序來處理所面對的問題。

正如同很多事物的學習過程，學習程式設計者往往一開始面對的是很多不知其所以然的指令語法，然是一個一個的範例，從模仿中學習，但除非學習動機或興趣特別強，很可能在產生興趣之前就已經被枯燥與挫折打敗。

如果能多增加一些背景知識，其實應該可以對學習者有些助益。學習程式設計所應具備的能力其實不是很多，但是在中學以前的課程中，並未被特別提出來，對於資訊工程的學生，往往在修了離散數學這門課時，才恍然大悟原來是怎麼回事。我們的目的在提出學習程式所需的背景知識與基本技巧，以便於增加在學習任何一種程式語言之前，對計算機原理與程式設計的方法能夠有所了解，而不致感到枯燥乏味。

大家應該要知道，指令語法不過是程式設計的枝微末節，在懂了原理之後，使用任何一種程式語言不過是套用它所規定的語法而已，

針對初學者來說，我們並不著重於程式的效率，而是著重於培養如何尋找出一個正確的計算機程序來處理問題，同時，我們也不著重於某個特定程式語言的語法，而只著重於觀念的建立。

chatGPT 一下...

學寫R程式的好處是什麼?

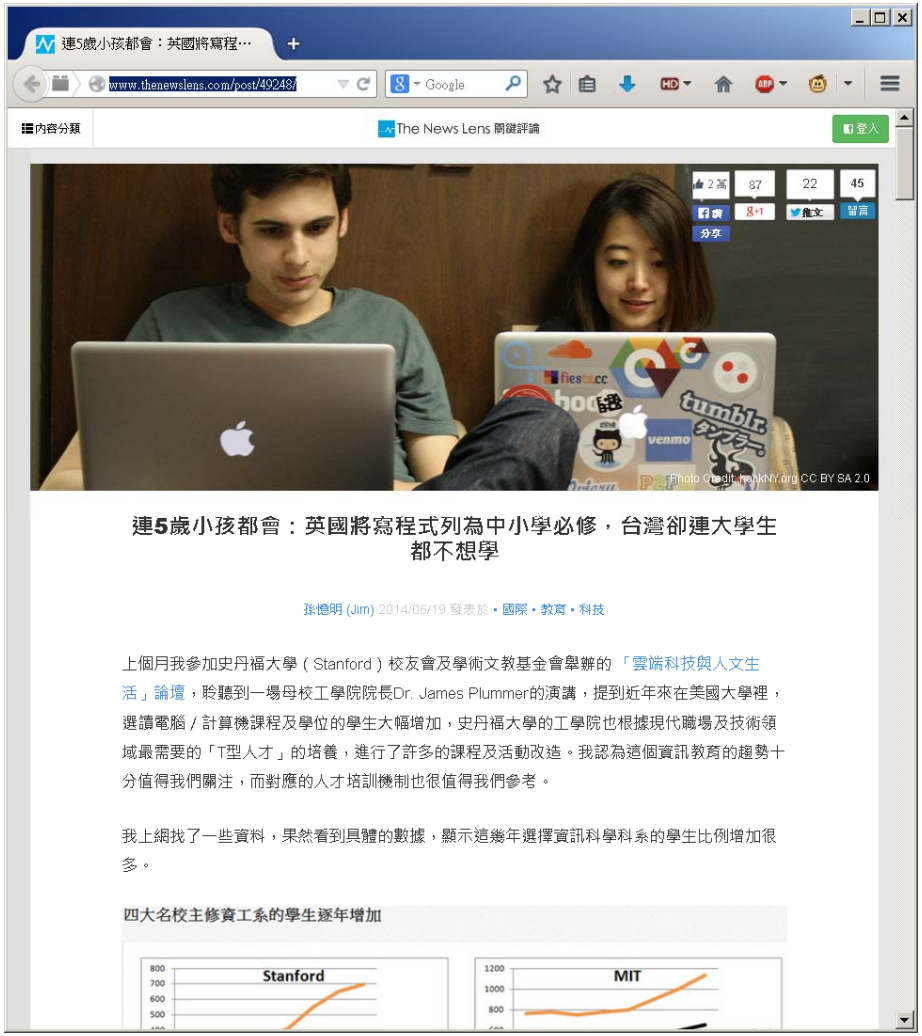
學寫R程式有以下好處：

1. 數據分析能力：R是專為數據分析設計的語言，學會R可以提高數據分析能力，包括資料讀取、資料整理、資料可視化、建立統計模型等。
2. 開源社群：R是一個開源程式語言，擁有一個龐大的社群，提供了大量的開源套件和資源，讓使用者能夠輕鬆地找到解決問題的方法。
3. 跨平台性：R可以在各種作業系統上運行，包括Windows、macOS和Linux等。
4. 高可擴性：R可以與其他程式語言（如Python和Java）進行整合，實現更複雜的功能。
5. 廣泛應用：R被廣泛應用於各種領域，包括生物醫學、金融、社會科學、工程學等，學會R可以幫助使用者應對各種數據分析問題。

本人教授R程式設計，希望給學生一些勉勵的話。

非常好，你選擇學習R程式是非常明智的決定。R程式是一個強大的工具，可以讓你在統計分析、資料視覺化和數據科學等領域中大顯身手。學習R程式需要時間和精力，但這是一個非常值得的投資，因為這將為你的未來事業和學術發展帶來無限可能。我相信，通過努力學習和實踐，你一定可以掌握R程式的精髓，成為一位優秀的數據科學家或統計分析師。

http://www.thenewslens.com/post/49248/
http://www.thenewslens.com/post/28632/



範例 1

輸入包含左右小括號之字串(最長爲40字元),
請判斷是否左右小括號配對正確。

(例1) 輸入: ((1+2)-3)*(4/5)
輸出: 括號配對正確。

(例3) 輸入: (((1+2+3)
輸出: 括號配對不正確。

(例3) 輸入: ((1+2)*(3+4)*(5+6))/(7+8)
輸出: 括號配對正確。

問題想法:

配對正確 (輸出)

↓

左括號個數 = 右括號個數

↓

計數字串「左，右」括號個數

↓

從字串中找出「左，右」括號

↓

字串由螢幕輸入

程式想法

從輸入字串中找出「左，右」括號

scan

從字串中，並找出所要的pattern

Pattern Matching and Replacement
`grep, sub, gsub, regexpr, gregexpr`

Substrings of a Character Vector
`substr, substring`

查 help, google,...

一般程式設計流程

1. 輸入格式應有一特別規定。 (例如一個點座標)
   - `x1 <- 3, y1 <- 2;  demo <- function(x1, y1){....}`
   - `x <- c(3, 2); demo <- function(x){....}`
2. 輸入/輸出測試OK。 (先給定Input，測試運算過程)
3. 加入判別。 (輸入型態, 參數範圍,長度大小, 真或偽)
4. 加入提示。 (互動程式, 給提示, 減少輸入錯誤)
5. 加入註解。 (便於日後維護)
6. 改變數名。 (有義意的名稱)
7. 群組，結構化。(重復的動作有哪一些?)
8. 寫作風格。 (格式，標頭，空格，對齊)
9. 好、巧、妙 。 (三境界)

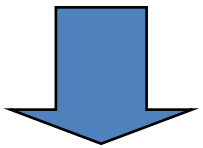
範例1: 實作(1)

程式

```
cat("第一題")
string <- "((1+2)*(3+4)*(5+6))/(7+8)"
gregexpr("[(]", string)[[1]]
length(gregexpr("[(]", string)[[1]])
```

執行

```
> cat("第一題")
第一題> string <- "((1+2)*(3+4)*(5+6))/(7+8)"
> gregexpr("[(]", string)[[1]]
[1]  1  2  8 14 21
attr(,"match.length")
[1] 1 1 1 1 1
> length(gregexpr("[(]", string)[[1]])
[1] 5
```



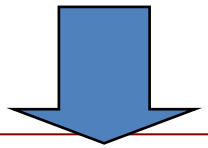
範例1: 實作(2)

程式

```
cat("第一題\n")
string <- "((1+2)*(3+4)*(5+6))/(7+8)"
left.num <- length(gregexpr("[(]", string)[[1]])
cat("left.num: ", left.num, "\n")
right.num <- length(gregexpr("[)]", string)[[1]])
cat("right.num: ", right.num, "\n")
if(left.num == right.num){
   cat("OK")
} else{
   cat("Not OK")
}
```

執行

```
> cat("第一題\n")
第一題
> string <- "((1+2)*(3+4)*(5+6))/(7+8)"
> left.num <- length(gregexpr("[(]", string)[[1]])
> cat("left.num: ", left.num, "\n")
left.num:  5
> right.num <- length(gregexpr("[)]", string)[[1]])
> cat("left.num: ", right.num, "\n")
left.num:  5
> if(left.num == right.num){
+    cat("OK")
+ } else{
+    cat("Not OK")
+ }
OK>
```



範例1: 實作(3)

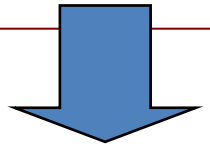
程式

```
ex1 <- function(){
    cat("第一題\n")

    #string <- "((1+2)*(3+4)*(5+6))/(7+8)"
    cat("輸入包含左右小括號之字串(最長為40字元)，請判斷是否左右小括號配對正確")
    string <- scan(what="character", nmax=1, quiet=TRUE)

    left.num <- length(gregexpr("[(]", string)[[1]])
    #cat("left.num: ", left.num, "\n")
    right.num <- length(gregexpr("[)]", string)[[1]])
    #cat("right.num: ", right.num, "\n")

    if(left.num == right.num){
       cat("OK")
    }else{
       cat("Not OK")
    }
}
```

執行

```
ex1()
```

```
> ex1()
第一題
輸入包含左右小括號之字串(最長為40字元)，請判斷是否左右小括號配對正確1: ((1+2)*(3+4)*(5+6))/(7+8)
OK>
```



範例1: 實作(4)

```
ex1 <- function(){
   cat("第一題\n")
   cat("########################################\n")
   cat("# 輸入包含左右小括號之字串(最長為40字元)，   #\n")
   cat("# 請判斷是否左右小括號配對正確                #\n")
   cat("# 例如輸入： {\tt ((1+2)-3)*(4/5)}       #\n")
   cat("########################################\n")

   ##輸入
   string <- scan(what = "character", nmax = 1, quiet = TRUE)

   ##找出"(" ")"，並計數
   left.num <- length(gregexpr("[(]", string)[[1]])
   right.num <- length(gregexpr("[)]", string)[[1]])

   ##判斷是否相等
   if(left.num == right.num){
     ##是的話，輸出OK
     cat("OK")
   }
   else{
     ##不是的話，輸出NOT OK
     cat("Not OK")
   }
}
```

繼續練習這一題(Y/N)

```
Y.or.N <- "y"

while(Y.or.N == "y"){
   ex1()
   cat("繼續練習這一題(Y/N): ")
   Y.or.N <- scan(what = "character", nmax = 1, quiet = TRUE)
   if(Y.or.N != "y" & Y.or.N != "n"){
      cat("輸入錯誤，再輸入一次 ")
   }
}
```

範例1: 完成

經過 n次修改及測試，存檔成ex1.R

```r
################################################################################
# Name: 範例主程式                                                             #
# Author: Han-Ming Wu                                                          #
# Date: 2008/11/05                                                             #
################################################################################

##############################
# ex1                        #
##############################
ex1 <- function(){
    cat("#########################################################\n")
    cat("# 第一題                                                #\n")
    cat("# 輸入包含左右小括號之字串(最長爲40字元)                 #\n")
    cat("# 請判斷是否左右小括號配對正確                           #\n")
    cat("# 例如輸入: ((1+2)-3)*(4/5)                             #\n")
    cat("#########################################################\n")

    repeat{

        ##輸入
        cat("請輸入包含左右小括號之字串(最長爲40字元): ")
        string <- scan(what="character", nmax=1, quiet=TRUE)
        if(nchar(string) < 40){

            ##找出"(" ")"，並計數
            left.num <- length(gregexpr("[(]", string)[[1]])
            right.num <- length(gregexpr("[)]", string)[[1]])
            cat("左小括號個數爲: ", left.num, "\n")
            cat("右小括號個數爲: ", right.num, "\n")

            ##判斷是否相等
            if(left.num== right.num){
               ##是的話，輸出"配對正確"
               cat("括號配對正確!\n")
            }
            else{
              ##不是的話，輸出"配對不正確"
              cat("括號配對不正確!\n")
            }
            break
        }
        else{
            cat("輸入錯誤!\n")
        }
    }
}
```

```r

##############################
# 是否繼續                   #
##############################
ask <- function(){
   cat("繼續練習這一題(y/n): ")
   Y.or.N <- scan(what="character", nmax=1, quiet=TRUE)
   return(Y.or.N)
}

##############################
# 檢查Input                  #
##############################
input.check <- function(answer){

    if(answer=="N" || answer=="n"){
      cat("謝謝練習，再會!\n")
    }
    else if(!(answer=="Y" || answer=="y")){
       cat("輸入錯誤!\n")
       Y.or.N <- ask()
       Y.or.N <- input.check(Y.or.N)
    }
    else{
       Y.or.N <- answer
   }
   return(Y.or.N)
}

##############################
# Example                    #
##############################
Y.or.N <- "y"
while(Y.or.N=="y" || Y.or.N=="Y"){
    ex1()
    Y.or.N <- ask()
    Y.or.N <- input.check(Y.or.N)
}
```

範例1: 執行

```
> source("ex1.R")
#########################################################
# 第一題                                                #
# 輸入包含左右小括號之字串(最長為40字元)                 #
# 請判斷是否左右小括號配對正確                           #
# 例如輸入: ((1+2)-3)*(4/5)                             #
#########################################################
請輸入包含左右小括號之字串(最長為40字元): 1:
((1+2)*(3+4)*(5+6))/(7+8)
左小括號個數為:  5
右小括號個數為:  5
括號配對正確!
繼續練習這一題(y/n): 1: y
#########################################################
# 第一題                                                #
# 輸入包含左右小括號之字串(最長為40字元)                 #
# 請判斷是否左右小括號配對正確                           #
# 例如輸入: ((1+2)-3)*(4/5)                             #
#########################################################
請輸入包含左右小括號之字串(最長為40字元): 1: (3+5)/(5*7*(9-3)))
左小括號個數為:  3
右小括號個數為:  4
括號配對不正確!
繼續練習這一題(y/n): 1: n
謝謝練習，再會!
```

範例 2

某國發行了1,5,10,50,100不同面額的鈔票, 若有人要從銀行領出N元,
銀行行員要如何發給鈔票, 使用的張數會最少?

(例) 輸入: 478
輸出:1元3張，5元1張，10元2張，50元1張，100元4張，共478元。

問題想法: 張數最少 → 面額最大的開始換

↓

剩下的錢，再換次大面額的

↓

重復，直到換完。

範例 3

平面上兩點(x_1, y_1), (x_2, y_2)之的距離式爲: $d = \sqrt{(x_1 - x_2)^2 + (y_1 - y_2)^2}$。
給定n個點$(n \leq 10)$, 找出構成最小周長的三角形的三個點。

(例) 輸入: (1,1)(0,0)(4,3)(2,0)(7,8)
輸出: 三點爲(1,1)(0,0)(2,0), 其周長爲4.828428。

問題想法:

三角形周長

⬇

三個點的兩兩距離

⬇

三個點是否成一個三角形?

⬇

10點任取三個點有多少可能?

⬇

跑完所有可能

⬇

取最小的周長的那一組

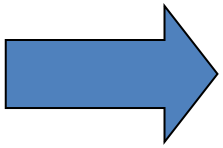
範例 3: 可能用到之副程式

```r
#
# 兩點之間距離
#
my.dis <- function(a, b){
    dis <- sqrt((a[1]-b[1])^2 + (a[2]-b[2])^2)
    return(dis)
}

#
# 判斷三點可否形成一個三角形
#
is.triangle <- function(a, b, c){

    is.tri <- FALSE
    dis.ab <- my.dis(a, b)
    dis.bc <- my.dis(b, c)
    dis.ac <- my.dis(a, c)
    count.1 <- ifelse(dis.ab + dis.bc > dis.ac, 0, 1)
    count.2 <- ifelse(dis.ab + dis.ac > dis.bc, 0, 1)
    count.3 <- ifelse(dis.bc + dis.ac > dis.ab, 0, 1)
    if(count.1 + count.2 + count.3 == 0){
       is.tri <- TRUE
    }
    return(is.tri)
}

```

```r
#
# 三角形之周長
#
my.perimeter <- function(a, b, c){

    if(is.triangle(a, b, c){
        length <- my.dis(a,b)+ my.dis(a,c)+ my.dis(b,c)
    }
    else{
        length <- Inf
    }
    return(length)
}

```

```r
for(i in 1: n){
   for(j in (i+1) : n){
       for(k in (j+1) : n){

           ....

       }
   }
}
```

找出規則
(重復、順序、整批)



寫成function
用apply, lappy, tapply,...等等

範例 4: 找出規律

輸入任何一個正整數 $n(n \leq 10)$, 輸出n階層的 Pascal 三角形。

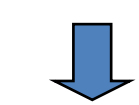
(例) 輸入: 5

輸出:

```
          1
        1   1
      1   2   1
    1   3   3   1
  1   4   6   4   1
```

問題想法:

觀察數列規則



細部修正(空格)

```
100     else if(x==6){
101                 cat("          1           \n")
102                 cat("        1   1         \n")
103                 cat("      1   2   1       \n")
104                 cat("    1   3   3   1     \n")
105                 cat("  1   4   6   4   1   \n")
106                 cat("1   5   10  10  5   1\n")
107                 }
```

choose(n, 0:n)

R Documentation

```
312 ############################################################
313 #                                                          #
314 #                                                          #
315 ############################################################
316 #' @title Box Plots For Interval Data
317 #' @description Produce box-and-whisker plot of the interval data
318 #' @param idata an IntervalData object
319 #' @details ...
320 #' @author Han-Ming Wu
321 #' @seealso boxplot.sbs.i, boxplotdou.i
322 #' @return The percentiles of the interval data
323 #' @examples
324 #' #' data(face)
325 #' idata.x <- face$x
326 #' y.C <- face$y
327 #' title <- "face data"
328 #' boxplot.i(idata.x)
329
330 boxplot.i <- function(idata, ...){
331
332
333     p
334     n
335     st
```

Exploratory Symbolic Data Analysis

Documentation for package 'exploreSDA' version 0.0.0.9000

- DESCRIPTION file.
- Code demos. Use demo() to run them.

Help Pages

boxplot.i	Box Plots For Interval Data
boxplot.sbs.i	The Side-by-Side Box Plots For Interval Data
cbind.h	Combine HistData Objects by Rows or Columns
cbind.i	Combine IntervalData Objects by Rows or Columns
get.subset	The Subset of the Histogram Data
hM.size	The Dimensions of the Histogram Data
plot.index.i	The Index Plot For Interval Data

boxplot.i {exploreSDA} R Documentation

Box Plots For Interval Data

Description

Produce box-and-whisker plot of the interval data

Usage

```
## S3 method for class 'i'
boxplot(idata, ...)
```

Arguments

idata an IntervalData object

Details

...

Value

The percentiles of the interval data

Author(s)

Han-Ming Wu

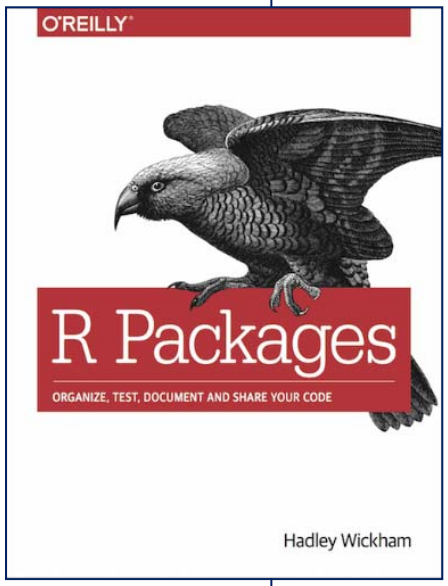
See Also

boxplot.sbs.i, boxplotdou.i

Examples

```
boxplot.i(idata.x)
```

[Package *exploreSDA* version 0.0.0.9000 Index]



http://r-pkgs.had.co.nz/

http://www.cc.ntu.edu.tw/chinese/epaper/0030/20140920_3006.html

Create an R Package in RStudio

https://www.youtube.com/watch?v=9PyQlbAEujY