# 第 11 章: 程式寫作

# 11: Programming

R 是一種 表達式 或 運算式語言 (expression language), 其任何一個述述句都可以看成是一個表達式或運算式, 它有指令形式與傳回結果的函式之運算, R 也是一種高階程式語言 (programming language), 因此提供了其它程序語言共有的條件 (if-else), 轉換 (switch), 迴圈 (loop) 等, 程序控制結構語法. R 是一種高階程式語言 (programming language), 因此如同 C 程式語言, R 語言可以寫作一些自定常用的 函式 (function).

## 11.1 控制結構語法 Control Structures

R 是一種表達式或運算式語言 (expression language), 其任何一個述述句都可以看成是一個表達式或運算式, 它有指令形式與傳回結果的函式之運算, 表達式或運算式可以續行, 只要前一行不是完整的表達式或運算式, 則下一行爲上一行的繼續. 表達式或運算式之間以 分號 ";" 作爲分隔或使用 換行 (new line) 作爲分隔.

R 也是一種高階程式語言 (programming language), 因此提供了其它程序語言共有的條件 (if-else), 轉換 (switch), 迴圈 (loop) 等, 程序控制結構語法. 許多個表達式或運算式可以放在一起, 組成一個更大的複合表達式或運算式, 且許多個指

令, 表達式或運算式可以用 “大括號” 包圍在一起, 例如,

```
> {expression1; ... ; expressionm}
```

此時, 這一組的複合表達式或運算式的結果, 是該組中最後一個指令的回傳的值. 既然一個組式是複合的表達式或運算式, 依然是一個達式或運算式, 它就可能放在其他括號中, 或放在一個更大的複合表達式或運算式中.

# 11.2 條件控制語言

# Conditionals Execution

R 是一種高階程式語言 (programming language), 因此提供了其它程序語言共有的 條件控制語言 (conditionals execution), 包含 條件控制語言 `if-else`, 條件控制函式 `ifelse()`, 以及 條件轉換 `switch()`, 等, 條件控制程序結構語法.

## 11.2.1 條件控制語言: if-else 敘述

R 程式語言的基本 條件控制語言 `if-else` 之語句形式爲

```
> if(condition) {
          ## condition = TRUE
          do expression1
    } else {
            ## condition = FALSE
            do expression2
      }

> if(condition1) {
          ## condition1 = TRUE
          do expression1
    } else if(condition2)  {
              ## condition1 = FALSE
              ## condition2 = TRUE
              do expression2
          } else {
                  ## condition2 = FALSE
                  do expression3
      }
```

若 表達式或運算式 很簡短, 則可寫成

```
> if (condition = TRUE) do expression1

> if (condition = TRUE) do expression1 else do expression2
```

其中 condition 是控制條件, condition 是一個邏輯操作, 判斷控制條件爲 “是” 或 “否” (TRUE or FALSE), condition 回傳一個 純量 (scalar), 若 condition 回傳 TRUE, 則執行 do expression1 之表達式或運算式; 若 condition 回傳 FALSE, 則 else 執行 do expression2 之表達式或運算式; 若無 do expression2, 則不進行任何運算, 回傳一個 NULL.

```
> ## if-else
> (x <- 1:5)
[1] 1 2 3 4 5
> if (x[3] > =  3) print("x is greater than or equal to 3")
[1] "x is greater than or equal to 3"
> if (x[3] > =  3) print("x > 3") else print("x < 3")
[1] "x > 3"
> if (x[2] > =  3) print("x is greater than or equal to 3")
> ## error: condition returns a logic vector
> if (x > 3) print("x > 3") else print("x < =  3")
[1] "x < =  3"
Warning message:
In if (x > 3) print("x > 3") else print("x < =  3") :
  the condition has length > 1 and only the first element will be used
```

在條件控制語言中的 運算式 (expression), expression1, expression2, expression3, ... 等等, 可以是用 “大括號” 包圍形成複合表達式或運算式. 一般寫成:

```
> if (condition1) {
        do expression21
        do expression22
        .......
        } else {
           do expression31
           do expression32
           ....... }
```

這樣的寫法可以使條件控制語言 if-else 不至於脫離前面的 if 語句.

```
> ## if-else {}
> (x <- 1:5)
[1] 1 2 3 4 5
> if (x[3] > =  3) {
         print("x is greater than 3")
```

```
6           print("or x is equal to 3")
7       } else{
8           print("x is less than 3")
9           print("x < 3")
10        }
11 [1] "x is greater than 3"
12 [1] "or x is equal to 3"
13 > #
14 > (x <- 1:5)
15 [1] 1 2 3 4 5
16 > if (x[2] > =  3) {
17          print("x is greater than 3")
18          print("or x is equal to 3")
19      } else{
20          print("x is less than 3")
21          print("x < 3")
22        }
23 [1] "x is less than 3"
24 [1] "x < 3"
```

條件控制語言 `if-else` 如其他函式, 可以回傳數值, `if-else` 可以用在向量所有元素都爲正數時才能做到的計算, 如計算對數值. 因此需要先檢查.

```
1 > ## if-else assign and calculation
2 > (x <- 0:4)
3 [1] 0 1 2 3 4
4 > if (any(x < =  0)) y <- log(0.0000001+x) else y <- log(x)
5 > y
6 [1] -16.1180957   0.0000001   0.6931472   1.0986123   1.3862944
7 > y <- if(any(x < =  0)) log(0.0000001+x) else log(x)
8 > y
9 [1] -16.1180957   0.0000001   0.6931472   1.0986123   1.3862944
```

條件控制語言 `if-else` 中 `if` 的控制條件 `condition` 若有許多條件必須同時考慮, 常用的方式是使用邏輯操作 `&&` (and) 與 `||` (or), 邏輯操作 `&&` (and) 與 `||` (or) 的指令如下:

```
1 > condition1 && condition2 && ...
2 > condition1  condition2  ...
```

上述每個條件 (condition) 是爲一個 純量 的邏輯值 (logical value), 當使用 `&&` 時, 僅有在 `condition1` 爲真的情況下, `condition2` 才執行判斷; 當使用 `||` 時, 僅有在 `condition1` 爲假的情況下, `condition2` 才執行判斷. 這裏要注意 `&` 和 `|` 將用於向量的所有元素, 而 `&&` 與 `||` 僅用於長度爲 1 的純量.

```
> ## if-else && and
> (x <- 2:5)
[1] 2 3 4 5
> if (all(x > 0) && all(log(x) > 0)) {
    y <- log(log(x));
    print(cbind(x, y));
    } else{
    cat('some x < =  1, unable to do log(log(x)) \n');
    }
     x           y
[1,] 2 -0.36651292
[2,] 3  0.09404783
[3,] 4  0.32663426
[4,] 5  0.47588500
> #
> ## if-else && and
> (x <- 0:3)
[1] 0 1 2 3
> if (all(x > 0) && all(log(x) > 0)) {
    y <- log(log(x));
    print(cbind(x, y));
    } else{
    cat('some x < =  1, unable to do log(log(x)) \n');
    }
some x < =  1, unable to do log(log(x))
```

條件控制語言 `if-else` 也可以是巢狀式的語法 (nested), 如

```
> if (condition1) {
      do expression1
      } else if (condition2) {
                  do expression2
                  } else if (condition3) {
                              do expression3
                              } else do expression4
```

控制條件敘述中, 若所有制條控制都傳回 `FALSE` 結果, 則執行 運算式-4 (`expression4`).

```
> ## if-else nested
> (x <- seq(from = 0.1, to = 0.9, by = 0.2))
[1] 0.1 0.3 0.5 0.7 0.9
> if (all(x > 1)) {
   print("all x > 1")
   } else if (all(log(x) > 0)) {
              print("all log(x) > 0")
              } else if (all(1/x < 0)) {
                          print("all 1/x < 0")
                         } else print("all are wrong")
[1] "all are wrong"
```

## 11.2.2 條件控制函式: ifelse()

R 是一個向量語言, 幾乎所有動作都是對向量進行的. 但 R 中的 條件控制語言 `if-else` 內的條件控制, 卻是一個少見的例外, 它的判斷條件是 純量, 僅有 `TRUE` 或 `FALSE`. 若想到利用 `if-else`, 對一個向量內的每個元素進行同一條件控制, 則無法使用 `if-else`. 例如, 對向量 $\underline{\mathbf{x}}$ (`x.vec`) 內的每個元素進行同一條件控制, 若 `x.vec > 0` 則 `x.vec = 1`, 其它 `x.dvec = 0`,

```
> if(x.vec>0) 1 else 0
```

但是當變數 `x.vec` 是一個向量時, 比較的結果也是一個向量, 這時條件控制語言 `if-else` 無法使用.

```
> ## if-else return a scalar
> (x <- seq(from = -2, to = 2))
[1] -2 -1  0  1  2
> if(x > 0) 1 else 0
[1] 0
Warning message:
In if (x > 0) 1 else 0 :
  the condition has length > 1 and only the first element will be used
```

所以這個運算式應該這樣寫

```
> y <- numeric(length(x))
> y[x > 0] <- 1
> y[x < =  0] <- 0
> y
[1] 0 0 0 1 1
```

R 提供了 `if-else` 條件控制語言的向量形式之函式 `ifelse()`. 函式指令爲

```
> ifelse(test, yes, no)
> ifelse(condition.vec, yes.vec, no.vec)
> (tmp <- yes; tmp[!test] <- no[!test]; tmp)
```

其中的主要引數分別爲:

- `test`, `condition.vec`: 條件測試向量, 回傳 logic vector.
- `yes`, `yes.vec`: 條件向量 `condition.vec` 內第 $i$ 個條件控制元素.

  - 若 test = TRUE 或 condition.vec[i] = TRUE, 回傳 yes 或 yes.vec[i] 對應元素.
  - 若 test = FALSE, 或 condition.vec[i] = FALSE 回傳 no 或 no.vec[i] 對應元素

- 函式 ifelse() 最終會傳回一個向量, 傳回的向量會和引數向量中長度最長的向量具有相同的長度.

```
> ## ifelse()
> (x <- seq(from = -2, to = 2))
[1] -2 -1  0  1  2
> ifelse(x > 0, 1, 0)
[1] 0 0 0 1 1
> #
> (x.vec <- seq(from = -2, to = 2))
[1] -2 -1  0  1  2
> (y.vec <- 1:5)
[1] 1 2 3 4 5
> (z.vec <- 11:15)
[1] 11 12 13 14 15
> ifelse(x.vec > 0, y.vec, z.vec)
[1] 11 12 13  4  5
> #
> ## ifelse() caution
> x <- c(6:-4)
> sqrt(x)  #- gives warning
 [1] 2.449490 2.236068 2.000000 1.732051 1.414214 1.000000 0.000000      NaN
 [9]      NaN      NaN      NaN
Warning message:
In sqrt(x) : NaNs produced
> sqrt(ifelse(x > =  0, x, NA))  # no warning
 [1] 2.449490 2.236068 2.000000 1.732051 1.414214 1.000000 0.000000       NA
 [9]       NA       NA       NA
> #
> ## Note: the following also gives the warning !
> ifelse(x > =  0, sqrt(x), NA)
 [1] 2.449490 2.236068 2.000000 1.732051 1.414214 1.000000 0.000000       NA
 [9]       NA       NA       NA
Warning message:
In sqrt(x) : NaNs produced
```

# 11.3 條件控制轉換函式: switch()

R 中有一個條件控制轉換函式 switch(), 可以利用條件回傳的 整數 或 文字 與其後續的選項對比, 並執行後續的選項內的運算式. 函式指令爲

```
> switch(EXPR, ...)
> switch(condition, expression1, expression3, ... , expression{k}, ... , expression{N})
```

其中的主要引數分別爲:

- 函式 switch() 內引數的數目包含 condition, expression1, expression2, ..., expressionN 共 $N+1$ 個引數.
- condition 是第一個引數, 做爲條件控制運算式, 並且回傳 整數 $k$ 或 字串.
- 若 condition 回傳一個 整數 $k$, 且若

$$\begin{cases} 1 \geq k \leq N, & \text{則執行 運算式 expressionk;} \\ k < 1 \text{ 或 } k > N, & \text{則回傳 NULL.} \end{cases}$$

- 若指令中的第一個引數 condition 回傳 字串, 則回傳之 字串, 將與其他引數名字的 字串 相配對, 若相配對吻合, 則執行 字串 與 名字 相配對吻合之 運算式.

```
> ## switch(integer, ...) integer
> x <- 3
> switch(x, 2+2, mean(1:10), rnorm(5))
[1] -1.4928635  1.0811194  0.5648061 -1.8008694 -0.5191843
> switch(2, 2+2, mean(1:10), rnorm(5))
[1] 5.5
> switch(6, 2+2, mean(1:10), rnorm(5))
```

```
> ## switch(character, ...) character
> (y <- rnorm(5))
[1] -0.7222697  0.5327645  0.7280864  2.2874758 -1.1042229
> test <- ("median")
> switch(test, mean = mean(y), median = median(y), sum = sum(y))
[1] 0.5327645
> switch("mean", mean = mean(y), median = median(y), sum = sum(y))
```

```
[1] 0.3443668
> switch("sum", mean = mean(y), median = median(y), sum = sum(y))
[1] 1.721834
```

## 11.4 重複控制語言 Repetitive Execution

R 是一種高階程式語言 (programming language), 因此提供了其它程序語言共有的 重複控制語言 (repetitive execution), 包含 迴圈控制 (loop control) `for()`, 判斷控制再重複執行 `while()`, 以及 重複執行再判斷控制 `repeat()` 與 `break` 等, 重複控制程序結構語法.

### 11.4.1 迴圈控制語言: for()

一般程式語言常用的 迴圈控制 (for loop control) 結構語法是 `for loop`, 在 R 中也提供了迴圈控制語言函式 `for()`, 可以對一個向量或列表的依序處理重複指令, `for()` 是指先指定數值並重複執行指令, 函式指令爲

```
> for(var in seq) expr
> for (id.name in sequelce.values) expression
```

其中的主要引數分別爲:

- `var`, `id.name` 是迴圈變數.
- `seq`, `sequence.values` 可以是一個向量 (vector) 或是一表列 (list), 通常是一個向量運算式 (常常以 `1:k` 這種形式出現).
- `expr`, `expression` 常常是根據迴圈變數 `id.name` 設計的成組運算式.
- 在 `id.name` 於 `sequence.values` 所有可以取到的數值時, 都會執行 `expr` 指令.

```
> ## for loop
> ## for()
> x <- 0
> for(i in 1:5){
      x <- x+i
      cat("i = ",i, ",", "x <- x+i  = ", x, "\n")
      }
i =  1 , x <- x+i = 1
i =  2 , x <- x+i = 3
i =  3 , x <- x+i = 6
i =  4 , x <- x+i = 10
i =  5 , x <- x+i = 15
> #
> (x <- 1:5)
[1] 1 2 3 4 5
> for(i in 1:length(x)){
      if(i > 1) {x[i] <- x[i-1] + x[i]}
      cat("i = ",i, ",", "x[i] <- x[i] + x[i-1] = ", x[i], "\n")
      }
i = 1 , x[i] <- x[i] + x[i-1] = 1
i = 2 , x[i] <- x[i] + x[i-1] = 3
i = 3 , x[i] <- x[i] + x[i-1] = 6
i = 4 , x[i] <- x[i] + x[i-1] = 10
i = 5 , x[i] <- x[i] + x[i-1] = 15
```

在 R 中使用 `for(i in 1:n)` 的計數迴圈時, 要避免一個常見錯誤, 即當 $n \leq 0$, `n` 爲 0 或負數時, 迴圈`1:n` 是一個從數值大到數值小的迴圈, 另外, 當 $n \leq 0$, `n` 爲 0 或負數時, 若不希望進入迴圈執行指令, 可以在迴圈外層, 設定判斷迴圈結束值是否小於開始值. 相較其他程式語言, R 語言裏面很少使用 `for()` 迴圈, 因 R 在每一迴圈產生中間之計算物件, 須等到迴圈結束時, 才會清除, 容易佔據記憶體, 或是程式停止執行, 使用 `apply()` 等函式, 執行速度與 `for()` 相當. 可以避免佔據記憶體, R 是一個向量語言, 儘量所有運算執行都是對向量直接進行. 使用 `for()` 迴圈時, 最好先設立所欲建立的向量或矩陣, 降低計算時間.

```
> ## for() speed
> x <- c()
> system.time(
    for(i in 1:40000){
      x <- c(x, i) # here i is combined with previous contents of x
    }
  )
   user  system elapsed
   2.07    0.03    2.12
> #
```

```
> x <- numeric(40000) #empty numeric vector
> system.time(
    for(i in 1:40000){
      x[i] <- i # changing value of particular element of x
    }
  )
   user  system elapsed
   0.05    0.00    0.05
```

## 11.4.2 迴圈控制語言: while()

迴圈控制語言 while() 是在開始執行前先判斷迴圈條件, 決定是否執行迴圈內程式指令. 函式指令爲

```
> while(condition) {expr}
```

其中的主要引數分別爲:

- condition 是第一個引數, 做爲判斷條件控制運算式, 並且回傳邏輯數值.
- 若條件控制 condition = TURE時, 則執行迴圈內運算式 expr.
- 並重複執行 運算式 expr, 直到當條件控制 condition = FALSE, 停止執行運算式 expr.
- while() 回傳運算式 expr 最後運算的結果.
- 若迴圈內運算式 expr 都未執行, 則迴圈控制 while() 回傳 NULL.
- while() 可與 next 或 code 並用.

迴圈控制 while() 常用來解決有關 迭代 (iteration) 的計算, 例如, 重複計算最大概似函數值與最大概似估計式, 直到收斂值爲止.

```
> ## while()
> (x <- 1:5)
[1] 1 2 3 4 5
> i <- 1
> while(i < =  5){
    if(i > 1) {x[i] <- x[i-1] + x[i]}
     cat("i = ",i, ",", "x[i] <- x[i] + x[i-1] = ", x[i], "\n")
```

```
      i <- i+1
      }
i = 1 , x[i] <- x[i] + x[i-1] = 1
i = 2 , x[i] <- x[i] + x[i-1] = 3
i = 3 , x[i] <- x[i] + x[i-1] = 6
i = 4 , x[i] <- x[i] + x[i-1] = 10
i = 5 , x[i] <- x[i] + x[i-1] = 15
> #
> ## while() + break x = 3
> x <- 1
> while(x < 5) {x <- x+1
                if (x = =  3) break
                print(x)
               }
[1] 2
> ## while() + skip x = 3
> x <- 1
> while(x < 5) {x <- x+1
                if (x = =  3) next
                print(x)
               }
[1] 2
[1] 4
[1] 5
```

## 11.4.3 迴圈控制語言: repeat 與 break

迴圈控制語言 repeat 與 while() 非常類似, repeat 可以重複執行指令, 不同於 while() 的是, repeat 在迴圈的最後才設定檢查迴圈控制條件, 通常與 break 並用, break 可以用於結束任何迴圈, break 是結束 repeat() 迴圈的唯一辦法. repeat 可與 next 並用, 且 next 可以用來跳脫特定的迴圈, 然後直接跳入 "下一次" 迴圈. repeated 迴圈控制語言指令爲

```
> repeat {expression1; ... ;  if (condition) break/next}
> repeat expr
  break
  next
```

其中的主要引數分別爲:

- 通常 repeat() 內 expr 是一個區塊 (block) 用 大括號 圍住, 至少含有二個

運算式, 一個爲執行運算計算式 expr, 與 一個爲條件控制運算式, 必須同時執行計算與檢查迴圈條件控制, condition.

- 若 (if) 符合特定迴圈的控制條件 condition, 則利用 break 結束 repeat() 迴圈.

```
> ## repeat()
> (x <- 1:5)
[1] 1 2 3 4 5
> i <- 1
> repeat{
     if(i > 1) {x[i] <- x[i-1] + x[i]}
      cat("i = ",i, ",", "x[i] <- x[i] + x[i-1] = ", x[i], "\n")
      i <- i+1
      if (i > 5) break
      }
i = 1 , x[i] <- x[i] + x[i-1] = 1
i = 2 , x[i] <- x[i] + x[i-1] = 3
i = 3 , x[i] <- x[i] + x[i-1] = 6
i = 4 , x[i] <- x[i] + x[i-1] = 10
i = 5 , x[i] <- x[i] + x[i-1] = 15
> #
> ## repeat{}
> x <- c("Hello","repeat")
> condit <- 2
> repeat {
    print(x)
    condit <- condit + 1
    if (condit > 5) {
      break
    }
  }
[1] "Hello"  "repeat"
[1] "Hello"  "repeat"
[1] "Hello"  "repeat"
[1] "Hello"  "repeat"
> ##
> repeat {
     if(i > 1) {x[i] <- x[i-1]+x[i]}
     cat("i = ",i, ",", "x[i] <- x[i]+x[i-1] = ", x[i], "\n")
     i <- i+1
     if (i > 5) break
     }
i =  1 , x[i] <- x[i]+x[i-1] =  1
i =  2 , x[i] <- x[i]+x[i-1] =  3
i =  3 , x[i] <- x[i]+x[i-1] =  6
i =  4 , x[i] <- x[i]+x[i-1] =  10
i =  5 , x[i] <- x[i]+x[i-1] =  15
```