Control Structures and Data I/O

Ting-Shuo Yo

October 7, 2016

基本控制結構

R 語言基本控制結構

- `if - else`
- `for`
- `while`
- `repeat` and `break`
- `next` and `return`

if - then

```
x <- 85
#
if(x > 90){            # if(<condition-1>){
  grade <- "A"         #     #do something
} else if(x > 80){     # } else if(<condition-2>) {
  grade <- "B"         #     #do something
} else {               # } else {
  grade <- "C"         #     #do something
}                      # }
#
print(c(x, grade))
```

```
## [1] "85" "B"
```

for loop

```r
for( x in 88:92){        # for(<condition>) { do-something }
  if(x > 90){
    grade <- "A"
  } else if(x > 80){
    grade <- "B"
  } else {
    grade <- "C"
  }
  print(c(x, grade))
}
```

```
## [1] "88" "B"
## [1] "89" "B"
## [1] "90" "B"
## [1] "91" "A"
## [1] "92" "A"
```

Nested for loops

`for` loops can be nested.

```
x <- matrix(1:6, 2, 3)

for(i in seq_len(nrow(x))) {
        for(j in seq_len(ncol(x))) {
                print(x[i, j])
        }
}
```

注意：大於三層以上的巢狀迴圈，會非常難以閱讀

while loop

While 迴圈從檢查一個邏輯敘述開始，如果是 true，才開始執行迴圈內容，執行完後，再重新檢查邏輯敘述，週而復始。

```
count <- 0
while(count < 10) {
        print(count)
        count <- count + 1
}
```

注意：While 的邏輯敘述要小心使用，以免陷入無窮迴圈。

while

有時候，我們會需要同時判斷多個狀況是否符合

```
z <- 5

while(z >= 3 && z <= 10) {
        print(z)
        coin <- rbinom(1, 1, 0.5)

        if(coin == 1) {  ## random walk
                z <- z + 1
        } else {
                z <- z - 1
        }
}
```

多個邏輯陳述的判斷一律是左邊的優先。

repeat

repeat 會啟動一個無窮迴圈 ，直到看到break才停下來。

```
x0 <- 1
tol <- 1e-8

repeat {
        x1 <- computeEstimate()

        if(abs(x1 - x0) < tol) {
                break
        } else {
                x0 <- x1
        }
}
```

repeat

- 使用 `repeat` 的風險，是如果偵測的狀況沒有發生，就永遠不會停下來。
  - 例如：數值方法一直沒有收斂。
- 比較好的方法，是放一個次數的限制，強迫迴圈停止。

next, return

在迴圈裡，我們有時候會想跳過某些情況，這時候`next`就派上用場了。

```
for(i in 1:100) {
        if(i <= 20) {
                ## Skip the first 20 iterations
                next
        }
        ## Do something here
}
```

`return` 會強迫迴圈停止並且傳回指定的資料。

Control Structures

小結

- R 主要的控制結構有 `if`, `while`, 和 `for` ，用來控制程式的運行和停止
- 「無窮迴圈」應該要極力避免，即使在理論上是正確的。
- 控制結構主要是用在「寫程式」，如果是在對話視窗裡工作，通常比較少用。（為什麼？）

資料輸入與輸出 I/O

讀取資料

R 讀取資料的常用函數：

- `read.table`, `read.csv`：讀取純文字的表格資料.
- `readLines`：讀取文字檔.
- `source`： 讀取並執行 R 程式檔 (`inverse` of `dump`).
- `dget`：讀取 R 程式檔 (`inverse` of `dput`).
- `load`：讀取 workspaces 檔（`*.RData`）.
- `unserialize`:讀取 R 物件檔.

輸出資料

R 輸出資料的常用函數：

- write.table
- writeLines
- dump
- dput
- save
- serialize

利用 read.table 函數讀取資料

`read.table` 是最常用的資料讀取函數之一，主要包含幾個參數:

- `file`：檔名（或檔案物件）
- `header`：T/F，檔案是否第一行包含欄位名稱
- `sep`：分隔資料欄位的符號
- `colClasses`：vector，欄位的資料型態
- `nrows`：要讀取資料的列數
- `comment.char`：註解符號
- `skip`：開頭要跳過的行數
- `stringsAsFactors`：T/F，要不要自動把文字資料轉換成 factor

read.table

在資料量不大的情況下，可以不加任何參數使用 `read.table`

```
data <- read.table("foo.txt")
```

R 會自動：

- 跳過開頭是 # 的資料列
- 計算欄位數量
- 偵測每個欄位的資料型態(但是有時會出錯)

加上參數，可以讓 R 的運行加快。

- `read.csv` 跟 `read.table` 基本上一樣,只是預設以逗號分隔欄位

用 read.table 讀取較大的資料檔

R 預設會把所有的資料讀到記憶體中，所以有些建議可以參考：

- 閱讀 `?read.table`
- 如果確定檔案裡沒有註解，設定 `comment.char = ""`
- 用 `colClasses` 指定欄位資料型態,可以加速到兩倍。
- 如果不知道欄位資料型態,可以先讀少數資料來判斷，例如：

```
initial <- read.table("datatable.txt", nrows = 100)
classes <- sapply(initial, class)
tabAll <- read.table("datatable.txt",
                     colClasses = classes)
```

- 指定 `nrows`，雖然不會加速讀取，但是可以節省記憶體。

以純文字型態輸出

- `dump` 和 `dput` 會以純文字型態輸出資料，可編輯，並且包含 meta-data，在資料損毀的時候比較可能復原。
- 純文字型態的輸出，可以和版本管理系統結合的較好，像是 `subversion` 或 `git`
- 純文字型態的輸出符合 “Unix philosophy”，可以跟其他命令列工具緊密結合
- 缺點：佔據比較大的儲存空間

dput / dget

我們可以用 dput 輸出 R 物件，並且以 dget 讀取

```
> y <- data.frame(a = 1, b = "a")
> dput(y)
structure(list(a = 1,
               b = structure(1L, .Label = "a",
                             class = "factor")),
          .Names = c("a", "b"), row.names = c(NA, -1L),
          class = "data.frame")
> dput(y, file = "y.R")
> new.y <- dget("y.R")
> new.y
   a  b
1  1  a
```

dump / source

我們可以用 `dump` 輸出多個 R 物件，並且以 `source` 讀取

```
> x <- "foo"
> y <- data.frame(a = 1, b = "a")
> dump(c("x", "y"), file = "data.R")
> rm(x, y)
> source("data.R")
> y
  a  b
1 1  a
> x
[1] "foo"
```

檔案物件

R 的資料讀取是透過 *connection* 物件，它可以是檔案或其他型態：

- `file`: 開啟本地檔案
- `gzfile`: 開啟 gzip 檔
- `bzfile`: 開啟 bzip2 檔
- `url`: 開啟網址

File Connections

```
> str(file)
function (description = "", open = "", blocking = TRUE,
          encoding = getOption("encoding"))
```

- `description`: 檔名
- `open` 可以用的參數包括
  - “r”: 唯獨
  - “w”: 寫入(並且刪除原有資料)
  - “a”: 延續寫入
  - “rb”, “wb”, “ab” reading, writing, or appending in binary mode

Connections

雖然 connections 是很好用的物件型態，但是實作上除非要對檔案本身作進階處理，很少會直接用。

```
con <- file("foo.txt", "r")
data <- read.csv(con)
close(con)
```

的效果跟下面的一行指令完全相同

```
data <- read.csv("foo.txt")
```

逐行讀取文字檔

```
> con <- gzfile("words.gz")
> x <- readLines(con, 10)
> x
 [1] "1080"      "10-point" "10th"      "11-point"
 [5] "12-point" "16-point" "18-point" "1st"
 [9] "2"         "20-point"
```

`writeLines` 函數會將一個文字 vector 的成員逐行的輸出到指定的檔案。

逐行讀取文字檔

`readLines` 在讀取網頁資料上也很方便：

```
## This might take time
con <- url("http://www.jhsph.edu", "r")
x <- readLines(con)
> head(x)
[1] "<!DOCTYPE HTML PUBLIC \"-//W3C//DTD HTML 4.0 Transitional//EN\">"
[2] ""
[3] "<html>"
[4] "<head>"
[5] "\t<meta http-equiv=\"Content-Type\" content=\"text/html;charset=utf-8
```

Quiz Time: Control and I/O

資料 data_quiz_1.csv 題組 Part 1

1. 請讀取 `data_quiz_1.csv`， 請問資料有幾欄？欄位名稱是什麼？
2. 將資料的前兩列輸出在 console 上，輸出長得怎麼樣?
3. 總共有多少筆觀測資料？
4. 將資料的最後兩列輸出在 console 上，輸出長得怎麼樣?
5. 第47筆觀測的 `Ozone` 值是多少？

解答 1

請讀取 `data_quiz_1.csv`， 請問資料有幾欄？欄位名稱是什麼？

```
data <- read.csv("../data_quiz_1.csv")
dim(data)
```

```
## [1] 153   6
```

```
names(data)
```

```
## [1] "Ozone"   "Solar.R" "Wind"    "Temp"    "Month"   "Day"
```

解答 2

將資料的前兩列輸出在 console 上，輸出長得怎麼樣?

```
head(data, 2)
```

```
##   Ozone Solar.R Wind Temp Month Day
## 1    41     190  7.4   67     5   1
## 2    36     118  8.0   72     5   2
```

解答 3

總共有多少筆觀測資料？

```
nrow(data)
```

```
## [1] 153
```

解答 4

將資料的最後兩列輸出在 console 上，輸出長得怎麼樣?

```
tail(data, 2)
```

```
##     Ozone Solar.R Wind Temp Month Day
## 152    18     131  8.0   76     9  29
## 153    20     223 11.5   68     9  30
```

解答 5

第47筆觀測的 Ozone 值是多少？

```
data[47,]
```

```
##    Ozone Solar.R Wind Temp Month Day
## 47    21     191 14.9   77     6  16
```

```
data$Ozone[47]
```

```
## [1] 21
```

資料 data_quiz_1.csv 題組 Part 2

1. `Ozone` 欄位有多少筆 missing data？
2. `Ozone` 欄位的平均值是多少？(請將 NA 值排除)
3. 選取資料中 `Ozone` 大於 31 而且 `Temp` 大於 90 的觀測，這些觀測的 `Solar` 欄位平均值是多少？
4. `Month` 是 6 的 `Temp` 平均值是多少？
5. 五月的 `Ozone` 最大值是多少？(`Month=5`)

解答 6

Ozone 欄位有多少筆 missing data？

```
sum(is.na(data$Ozone))
```

```
## [1] 37
```

```
sum(!complete.cases(data$Ozone))
```

```
## [1] 37
```

解答 7

Ozone 欄位的平均值是多少？(請將 NA 值排除)

```
apply(data, 2, mean)
```

```
##     Ozone   Solar.R      Wind      Temp     Month       Day
##        NA        NA  9.957516 77.882353  6.993464 15.803922
```

```
mean(data$Ozone, na.rm=TRUE)
```

```
## [1] 42.12931
```

解答 8

選取資料中 Ozone 大於 31 而且 Temp 大於 90 的觀測，這些觀測的 Solar 欄位平均值是多少？

```
subdata <- subset(data, data$Ozone>31 & data$Temp>90, select=Solar.R)
apply(subdata, 2, mean)
```

```
## Solar.R
##   212.8
```

```
mean(data$Solar.R[which(data$Ozone>31 & data$Temp>90)], na.rm=T)
```

```
## [1] 212.8
```

解答 9

Month 是 6 的 Temp 平均值是多少？

```
subdata <- subset(data, data$Month==6, select=Temp)
apply(subdata, 2, mean)
```

```
## Temp
## 79.1
```

```
mean(data$Temp[which(data$Month==6)], na.rm=T)
```

```
## [1] 79.1
```

解答 10

五月的 Ozone 最大值是多少？(Month=5)

```
subdata <- subset(data, data$Month==5, select=Ozone)
max(subdata, na.rm=TRUE)
```

```
## [1] 115
```

```
max(data$Ozone[which(data$Month==5)], na.rm=T)
```

```
## [1] 115
```