# 第 10 天

## 自訂函數

我們在**函數**章節學習了很多 R 語言針對數值、文字與描述性統計的內建函數，既然函數對我們的重要性這麼高，您有沒有想過自己撰寫函數來滿足特殊的需求呢？比如說，針對手邊雜亂無章的資料，撰寫一個叫做 clean\_data() 函數，只要呼叫它，就能夠回傳您想要的資料樣式。聽起來真是令人心動，我們趕快來瞭解一下如何自訂函數。

### 自訂函數的外觀架構

一個自訂函數的外觀架構長得像這個樣子：

function\_name <- function(輸入1, 輸入2, ..., 輸入 n, 參數1, 參數2, ..., 參數 n){  
 # 呼叫函數後執行的程式  
 return(輸出)  
}

首先要給函數取個名字（function\_name），告訴 R 語言這是一個函數的物件（function(...)），接著在小括號中放入想好的輸入（inputs）與參數（parameters），然後在大括號內縮排撰寫我們主要的程式，最後把輸出結果放在 return() 函數裡面。

我們很快地來實作第一個自訂函數，它的作用是將輸入的數字平方之後回傳，我們就叫它 my\_squared() 函數。記得在撰寫多行且具有縮排的程式時候，我們都應該使用左上角的來源（Source）區塊編寫比較方便：

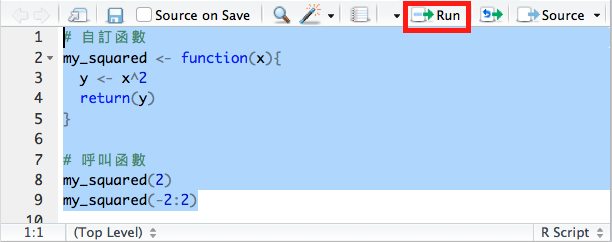


圖 10-1 my\_squared() 函數

> # 自訂函數  
> my\_squared <- function(x){  
+ y <- x^2  
+ return(y)  
+ }  
>   
> # 呼叫函數  
> my\_squared(2)  
[1] 4  
> my\_squared(-2:2)  
[1] 4 1 0 1 4

### 練習更多自訂函數

熟能生巧，我們再來練習自訂幾個簡單的函數！

#### 計算圓的面積

我們來實作一個自訂函數叫做 circle\_area()，當使用者輸入一個圓形的半徑，它會計算出該圓形的面積。

> # 自訂函數  
> circle\_area <- function(r){  
+ area <- pi \* r^2 # R 語言有內建圓周率 pi  
+ return(area)  
+ }  
> # 呼叫函數  
> circle\_area(3)  
[1] 28.27433  
> circle\_area(5)  
[1] 78.53982

#### 計算圓的周長

我們來實作一個自訂函數叫做 circle\_circum()，當使用者輸入一個圓形的半徑，它會計算出該圓形的周長。

> # 自訂函數  
> circle\_circum <- function(r){  
+ circum <- 2 \* pi \* r # R 語言有內建圓周率 pi  
+ return(circum)  
+ }  
> # 呼叫函數  
> circle\_circum(3)  
[1] 18.84956  
> circle\_circum(5)  
[1] 31.41593

#### 讓使用者決定要計算圓面積或周長

我們把前面兩個自訂函數的功能整合起來到一個函數裡面，這個時候我們會正式納入**參數（parameters）**的觀念到這個自訂函數中，使用一個邏輯值參數 is\_area 來讓使用者呼叫時決定要計算面積或者周長。

> # 自訂函數  
> circle\_calculator <- function(r, is\_area){  
+ area <- pi \* r^2  
+ circum <- 2 \* pi \* r  
+ if (is\_area == TRUE){  
+ return(area)  
+ } else {  
+ return(circum)  
+ }  
+ }  
> # 呼叫函數  
> circle\_calculator(3, is\_area = TRUE)  
[1] 28.27433  
> circle\_calculator(3, is\_area = FALSE)  
[1] 18.84956

這裡我們設計使用邏輯值做為 is\_area 參數的原因是預期僅有兩種可能（面積或周長），假如使用者呼叫函數時沒有指定 is\_area 參數，就會出現錯誤訊息：

> # 自訂函數  
> circle\_calculator <- function(r, is\_area){  
+ area <- pi \* r^2  
+ circum <- 2 \* pi \* r  
+ if (is\_area == TRUE){  
+ return(area)  
+ } else {  
+ return(circum)  
+ }  
+ }  
> # 呼叫函數  
> circle\_calculator(3)  
Error in circle\_calculator(3) :   
 argument "is\_area" is missing, with no default

於是我們稍微修改一下自訂函數，給 is\_area 一個**預設值**為 TRUE，意思就是說當使用者呼叫函數時沒有指定 is\_area 參數時，函數會指派 TRUE 給該參數（假定使用者是計算圓面積。）

> # 自訂函數  
> circle\_calculator <- function(r, is\_area = TRUE){  
+ area <- pi \* r^2  
+ circum <- 2 \* pi \* r  
+ if (is\_area == TRUE){  
+ return(area)  
+ } else {  
+ return(circum)  
+ }  
+ }  
> # 呼叫函數  
> circle\_calculator(3)  
[1] 28.27433

#### 有多個輸出的自訂函數

還記得我們在**把變數集結起來（3）**學習清單這個資料結構的時候，我們就舉了 R 語言的 lm() 函數作為例子，一個線性模型可以包含很多的輸出（像是係數、殘差…等），而眾多的輸出可以加以命名，並儲存在清單之中。

因此我們可以再修正一下 circle\_calculator() 這個自訂函數，這次我們將圓面積與周長包含在一個清單中輸出，讓使用者依照名稱選擇圓面積或者周長：

> # 自訂函數  
> circle\_calculator <- function(r){  
+ area <- pi \* r^2  
+ circum <- 2 \* pi \* r  
+ ans\_list <- list(area = area, circum = circum) # 為清單中的物件命名方便用 $ 取用  
+ return(ans\_list)  
+ }  
> # 呼叫函數  
> circle\_with\_r\_3 <- circle\_calculator(3)  
> circle\_with\_r\_3$area  
[1] 28.27433  
> circle\_with\_r\_3$circum  
[1] 18.84956

#### 練習實作 length() 函數

我們可以利用 R 語言的內建函數 length() 計算一個向量的長度（元素個數），我們可以試著自訂 my\_length() 函數做一樣的事情：

> # 自訂函數  
> my\_length <- function(x){  
+ counts <- 0  
+ for (i in x){  
+ counts <- counts + 1  
+ }  
+ return(counts)  
+ }  
>   
> # 建立一個向量  
> my\_vec <- seq(from = 15, to = 87, by = 3)  
>   
> # 呼叫內建與自訂函數  
> length(my\_vec)  
[1] 25  
> my\_length(my\_vec)  
[1] 25

我們初始化一個計數器 counts，每一次迭代時，這個計數器就會累加 1，當迭代完畢後，回傳計數器最後的值。

#### 練習實作 sum() 函數

我們可以利用 R 語言的內建函數 sum() 計算一個向量的元素總和，我們可以試著自訂 my\_sum() 函數做一樣的事情：

> # 自訂函數  
> my\_sum <- function(x){  
+ summation <- 0  
+ for (i in x){  
+ summation <- summation + i  
+ }  
+ return(summation)  
+ }  
> # 呼叫內建與自訂函數  
> sum(1:10)  
[1] 55  
> my\_sum(1:10)  
[1] 55

### 處理一個雜亂無章的資料

回應我們今天一開始提到的概念，手邊有一個雜亂無章的資料，如果能寫一個 clean\_data() 函數把它依照自己理想的方法處理，是多麽輕鬆方便的一件事呢？假設我們雜亂無章的資料長得像這樣：

> # 雜亂無章的資料  
> messy\_data <- data.frame(c(1, 2, 3, 4, NA), c(1, 2, 3, NA, 5), c(1, 2, NA, 4, 5))  
> names(messy\_data) <- c("a", "b", "c")  
> messy\_data  
 a b c  
1 1 1 1  
2 2 2 2  
3 3 3 NA  
4 4 NA 4  
5 NA 5 5

我們理想中的 clean\_data() 函數會提供兩種輸出：

1. 把有出現 NA 的觀測值都刪除
2. 把 NA 用某個數值取代

# 自訂函數  
clean\_data <- function(df, impute\_value){  
 n\_rows <- nrow(df)  
 na\_sum <- rep(NA, times = n\_rows)  
 for (i in 1:n\_rows){  
 na\_sum[i] <- sum(is.na(df[i, ])) # 計算每個觀測值有幾個 NA  
 df[i, ][is.na(df[i, ])] <- impute\_value # 把 NA 用某個數值取代  
 }  
 complete\_cases <- df[as.logical(!na\_sum), ] # 把沒有出現 NA 的觀測值保留下來  
 imputed\_data <- df  
 df\_list <- list(complete\_cases = complete\_cases, imputed\_data = imputed\_data)  
 return(df\_list)  
}

讓我們來呼叫 clean\_data() 函數，並且指定用 999 取代 NA 試試看：

> # 自訂函數  
> clean\_data <- function(df, impute\_value){  
+ n\_rows <- nrow(df)  
+ na\_sum <- rep(NA, times = n\_rows)  
+ for (i in 1:n\_rows){  
+ na\_sum[i] <- sum(is.na(df[i, ])) # 計算每個觀測值有幾個 NA  
+ df[i, ][is.na(df[i, ])] <- impute\_value # 把 NA 用某個數值取代  
+ }  
+ complete\_cases <- df[as.logical(!na\_sum), ] # 把沒有出現 NA 的觀測值保留下來  
+ imputed\_data <- df  
+ df\_list <- list(complete\_cases = complete\_cases, imputed\_data = imputed\_data)  
+ return(df\_list)  
+ }  
> # 雜亂無章的資料  
> messy\_data <- data.frame(c(1, 2, 3, 4, NA), c(1, 2, 3, NA, 5), c(1, 2, NA, 4, 5))  
> names(messy\_data) <- c("a", "b", "c")  
> # 呼叫函數  
> cleaned\_data <- clean\_data(messy\_data, impute\_value = 999)  
> cleaned\_data$complete\_cases  
 a b c  
1 1 1 1  
2 2 2 2  
> cleaned\_data$imputed\_data  
 a b c  
1 1 1 1  
2 2 2 2  
3 3 3 999  
4 4 999 4  
5 999 5 5

太完美了，不論是將有遺漏值的觀測值全部刪去或是替換成指定的數字都沒有問題！

### 小結

好啦！第十天的內容就是這麼多，我們學會了如何自訂函數，並且透過大量的範例熟練輸入、參數與輸出的設計，最後我們設計了一個 clean\_data() 函數可以針對資料框中的遺漏值做兩種不同的處理。

### 練習題

###### 1. 請練習自訂一個函數 my\_factorial()，只要輸入整數 n，就會計算出 回傳

# 自訂函數  
my\_factorial <- function(n){  
 n <- as.integer(\_\_\_)  
 ans <- 1  
 for (i in 1:n){  
 ans <- \_\_\_ \* \_\_\_  
 }  
 return(ans)  
}  
  
# 呼叫函數  
my\_factorial(5)

###### 2. 延續我們實作的 my\_sum() 函數與 my\_length() 函數，請練習自訂一個函數 my\_mean()，平均數的公式：

# 自訂函數  
my\_mean <- function(x){  
 my\_sum <- 0  
 my\_length <- 0  
 for (i in x){  
 my\_sum <- \_\_\_ + \_\_\_  
 my\_length <- \_\_\_ + \_\_\_  
 }  
 return(\_\_\_ / \_\_\_)  
}  
  
# 呼叫函數  
my\_mean(1:10)  
mean(1:10)