# 成為初級資料分析師 I R 程式設計與資料科學應用

*程式封裝：自訂函數*

**郭耀仁**

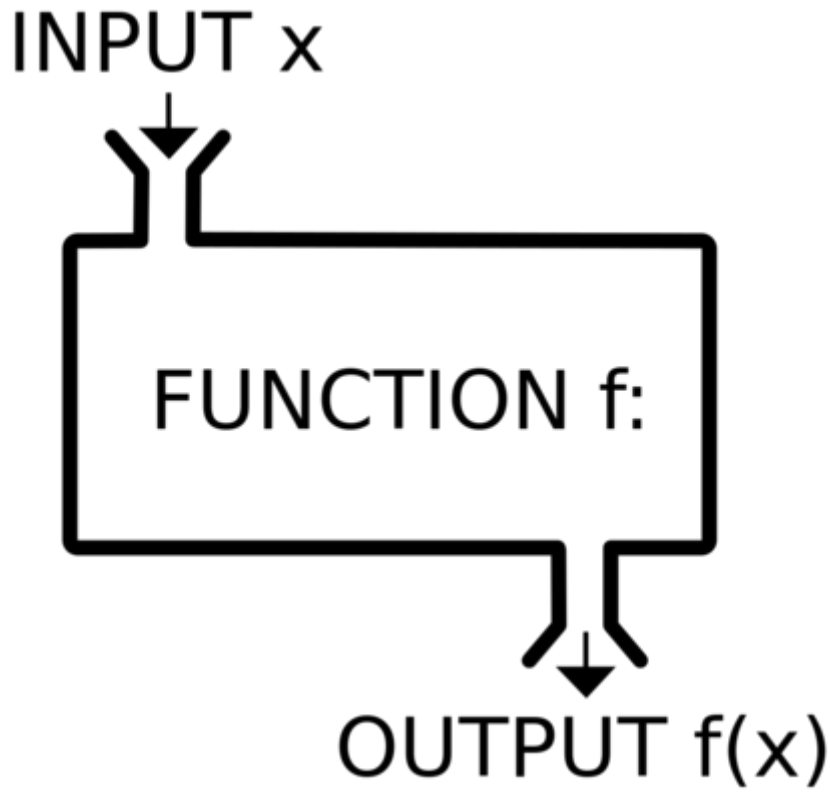
*The way R works is pretty straightforward, you apply functions to objects.*

*Greg Martain*

# 大綱

- 關於函數
- 常用的 R 內建函數
- 自訂函數
- 全域（Global）與區域（Local）
- 向量化函數
- 遞迴（Recursion）

# 關於函數


INPUT x
FUNCTION f:
OUTPUT f(x)

# 函數由四個元件組合而成

- 輸入 INPUTS
- 參數 ARGUMENTS
- 主體 BODY
- 輸出 OUTPUTS

# 常用的 R 內建函數

# 内建数值函数

# abs()

取絕對值

In [ ]:
```
abs(-5566)
abs(5566)
```

# sqrt()

平方根

In [ ]:
```
sqrt(2)
sqrt(3)
```

# ceiling()

無條件進位

In [ ]:
```
ceiling(sqrt(2))
ceiling(sqrt(3))
```

# floor()

無條件捨去

In [ ]:
```
floor(sqrt(2))
floor(sqrt(3))
```

# round()

四捨五入

In [ ]:
```
round(sqrt(2))
round(sqrt(3))
```

# exp()

指数函数

$$exp(x) = e^x \text{ , where } e = 2.71828...$$

In [ ]:
```
exp(1)
exp(2)
```

# log()

以 $e$ 為底的對數函數

In [ ]:
```
e = exp(1)
ee = exp(2)
log(e)
log(ee)
```

# log10()

以 10 為底的對數函數

In [ ]:
```
log10(10)
log10(10**2)
```

# 內建描述性統計函數

# mean()

平均数

In [ ]:
```
mean(1:10)
```

# sd()

標準差

In [ ]:
```
sd(1:10)
```

# median()

中位数

In [ ]:
```
median(1:9)
```

# range()

最大值與最小值

In [ ]:
```
range(11:100)
```

# sum()

加總

In [ ]:
```
sum(1:100)
```

# max()

最大値

In [ ]:
```
max(11:100)
```

# min()

最小値

In [ ]:
```
min(11:100)
```

# 內建文字處理函數

# unique()

取獨一值

In [ ]:
```
cities <- c("New York", "Boston", "Tokyo", "Kyoto", "Taipei")
countries <- c("United States", "United States", "Japan", "Japan", "Taiwan")
unique(cities)
unique(countries)
```

## toupper()

轉換為大寫

In [ ]:
```
toupper("Luke, use the Force!")
```

# tolower()

轉換為小寫

In [ ]:
```
tolower("Luke, use the Force!")
```

# substr()

擷取部分字串

In [ ]:
```
luke <- "Luke, use the Force!"
substr(luke, start = 1, stop = 4)
substr(luke, start = 11, stop = nchar(luke))
```

# grep()

回傳指定特徵有出現的索引值

In [ ]:
```
avengers <- c("The Avengers", "Avengers: Age of Ultron", "Avengers: Infinity War",
"Avengers: Endgame")
grep(pattern = "Avengers", avengers)
grep(pattern = "Endgame", avengers)
```

# sub()

取代指定特徵

In [ ]:
```
skywalker <- "Anakin Skywalker"
sub(pattern = "Anakin", replacement = "Luke", skywalker)
```

# strsplit()

依據指定特徵分割文字

In [ ]:
```
avengers <- c("The Avengers", "Avengers: Age of Ultron", "Avengers: Infinity War",
"Avengers: Endgame")
strsplit(avengers, split = " ")
strsplit(avengers, split = ":")
```

# paste() 與 paste0()

連接文字

In [ ]:
```
paste("Avengers:", "Endgame")
paste0("Avengers:", "Endgame")
```

# trimws()

消除前（leading）或後（trailing）的空白

In [ ]:
```
luke <- "     Luke, use the Force!     "
trimws(luke)
trimws(luke, which = "left")
trimws(luke, which = "right")
```

# 自訂函數

# 在自訂函數時考慮 6 個元件

- 函數名稱（命名風格與物件命名風格相同，儘量使用動詞）
- 輸入的命名與設計
- 參數的命名、設計與預設值
- 函數主體
- 輸出的命名與設計
- 使用保留字 `function` 與 `return()`

# 自訂函數的 Code Block

```
FUNCTION_NAME <- function(INPUTS, ARGUMENTS, ...) {
  # BODY
  return(OUTPUTS)
}
```

# 單一輸入的函數

將攝氏溫度轉換為華氏溫度的函數 `celsius_to_fahrenheit()`

$$Fahrenheit_{(^\circ F)} = Celsius_{(^\circ C)} \times \frac{9}{5} + 32$$

In [ ]:
```
celsius_to_fahrenheit <- function(x) {
    return(x*9/5 + 32)
}
celsius_to_fahrenheit(20)
```

# 隨堂練習：將公里轉換為英里的函數 `km_to_mile()`

$$Miles = Kilometers \times 0.62137$$

In [2]:
```
km_to_mile(21.095)
km_to_mile(42.195)
```

13.10780015

26.21870715

# 隨堂練習：判斷輸入 x 是否為質數的函數 `is_prime()`

In [4]:
```
is_prime(1)
is_prime(2)
```

FALSE

TRUE

# 兩個以上輸入的函數

In [ ]:
```
get_bmi <- function(height, weight) {
    return(weight / (height*0.01)**2)
}
get_bmi(191, 91)
```

# 隨堂練習：回傳介於 x 與 y 之間的質數個數（包含 x 與 y 如果他們也是質數）的函數 `count_primes()`

In [6]:
```
count_primes(1, 5)
count_primes(9, 19)
```

3

4

# 參數有預設值的函數

In [ ]:
```
temperature_converter <- function(x, to_fahrenheit = TRUE) {
    if (to_fahrenheit) {
        return(x*9/5 + 32)
    } else {
        return((x - 32)*5/9)
    }
}
temperature_converter(20)
temperature_converter(68, to_fahrenheit = FALSE)
```

# 隨堂練習：計算圓面積或圓周長的函數 `circle_calculator()`（預設計算圓面積）

In [8]:
```
circle_calculator(3)
circle_calculator(3, is_area = FALSE)
```

28.2743338823081

18.8495559215388

# 多個輸出的函數

In [ ]:
```
get_bmi_and_label <- function(height, weight) {
    bmi <- weight / (height/100)**2
    if (bmi > 30) {
        label <- "Obese"
    } else if (bmi < 18.5) {
        label <- "Underweight"
    } else if (bmi > 25) {
        label <- "Overweight"
    } else {
        label <- "Normal weight"
    }
    bmi_and_label <- list(
        bmi = bmi,
        bmi_label = label
    )
    return(bmi_and_label)
}
```

In [ ]:
```
get_bmi_and_label(216, 147) # Shaquille O'Neal
get_bmi_and_label(203, 113) # LeBron James
get_bmi_and_label(191, 82) # Steve Nash
get_bmi_and_label(231, 91) # Manute Bol
```

# 隨堂練習：回傳介於 x 與 y 之間的質數個數（包含 x 與 y 如果他們也是質數）以及質數明細的函數 `get_primes_and_counts()`

In [10]:
```
get_primes_and_counts(1, 5)
get_primes_and_counts(9, 19)
```

**$primes**

2 3 5

**$prime_counts**

3

**$primes**

11 13 17 19

**$prime_counts**

4

# 全域（Global）與區域（Local）

# 什麼是全域與區域?

- 在函數的 Code Block 以外所建立的物件屬於全域
- 在函數的 Code Block 中建立的物件屬於區域

# 全域物件與區域物件的差別?

- 區域物件**僅可**在區域中使用
- 全域物件可以在全域以及區域中使用

In [ ]:

```
# 區域物件僅可在區域中使用
get_sqrt <- function(x) {
    sqrt_x <- x**0.5
    return(sqrt_x)
}

get_sqrt(2)
sqrt_x # Local object cannot be accessed in global
```

In [ ]:
```
# 全域物件可以在全域以及區域中使用
x <- 2
sqrt_x <- x**0.5
sqrt_x # Global object can be accessed in global, of course
```

In [ ]:

```
# 全域物件可以在全域以及區域中使用
x <- 2
sqrt_x <- x**0.5

get_sqrt <- function() {
    return(sqrt_x) # Global object can be accessed in local
}
get_sqrt()
```

# 向量化函數

# 什麼是向量化函數?

利用 A 函數將 B 函數**向量化**至某個資料結構的方法，其中 A 函數是特定的函數型函數（Functional Functions）

In [ ]:

```
# 將一個 list 中的每個數字都平方
my_list <- list(
    11,
    12,
    13,
    14,
    15
)
my_list
```

# 老實說，我們會有個衝動這樣寫

In [ ]:
```
my_list**2
```

## 注意這是一個 `list` 而不是向量

In [ ]:
```
for (i in my_list) {
    print(i**2)
}
```

# 使用 lapply() 函數將 get_squared() 函數向量化至 my_list 之上

In [ ]:

```
get_squared <- function(x) {
    return(x**2)
}

lapply(my_list, FUN = get_squared)
```

## 搭配匿名函數（Anonymous Function）將平方運算向量化至 my_list 之上

In [ ]:
```
lapply(my_list, FUN = function(x) return(x**2))
```

# R 常用的函數型函數（functional functions）

- `lapply()`
- `sapply()`
- `apply()`
- `mapply()`

## `sapply()`

- Simplified apply
- 回傳向量而非 `list`

In [ ]:
```
sapply(my_list, FUN = function(x) return(x**2))
```

# apply()

將函數向量化至二維的資料結構（matrix 與 data.frame）

In [ ]:
```
my_matrix <- matrix(1:12, nrow = 2)
my_matrix
apply(my_matrix, MARGIN = 1, FUN = sum)
apply(my_matrix, MARGIN = 2, FUN = sum)
```

# mapply()

將具有多個輸入的函數向量化

In [ ]:
```
weights <- list(91, 82, 113, 147)
heights <- list(231, 191, 203, 216)
bmis <- mapply(FUN = function(h, w) return(w/(h*0.01)**2), heights, weights)
bmis
```

# 隨堂練習：使用向量化函數將 5 個球員的姓氏（last name）擷取出來並轉換成大寫

In [11]:
```
fav_players <- list("Steve Nash", "Paul Pierce", "Dirk Nowitzki", "Kevin Garnett",
"Hakeem Olajuwon")
```

In [13]:
```
ans
```

1. 'NASH'
2. 'PIERCE'
3. 'NOWITZKI'
4. 'GARNETT'
5. 'OLAJUWON'

# 遞迴（Recursion）

# 什麼是遞迴（Recursion）？

在一個函數中呼叫函數本身的技法


Source: https://twitter.com/ProgrammersMeme/status/1147050956821008384 (https://twitter.com/ProgrammersMeme/status/1147050956821008384)

# 階乗（factorial）の計算

$$n! = 1 \times 2 \times 3 \times \ldots \times n$$

In [ ]:

```
factorial <- function(n) {
    if (n == 1) {
        return(n)
    } else {
        return(n * factorial(n-1))
    }
}
factorial(1)
factorial(2)
factorial(3)
```

# 隨堂練習： 建立 fibonacci 數列

$$F_0 = 0, F_1 = 1$$

$$F_n = F_{n-1} + F_{n-2} \text{ , For } n > 1$$

In [15]:
```
fibonacci(1)
fibonacci(2)
fibonacci(20)
```

```
0

0 1

0 1 1 2 3 5 8 13 21 34 55 89 144 233 377 610 987 1597 2584
4181
```