Unit2-1 R 語言資料語法

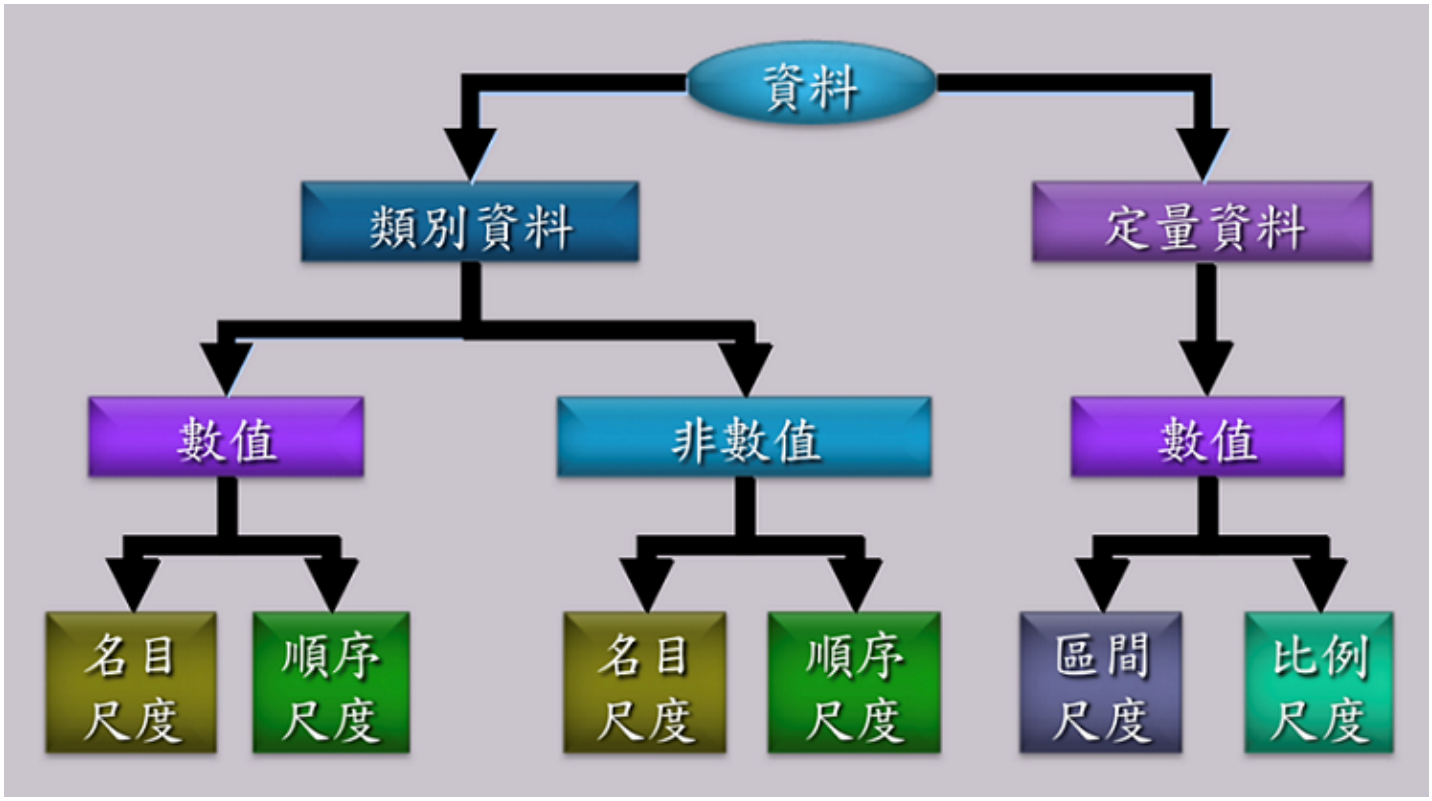
Table of Content

- 矩陣的運算
- 解線性聯立方程組

1. R語言的變數類型

通常，資料集的列表示觀察值(受訪者，individuals)，行表示變數(variables)，為觀察值的特性，而資料集中的元素有各種不同的類型。對於不同變數元素資料的種類，統計學依據其尺度水平劃分為：名目尺度（nominal）、順序尺度（ordinal）、等距尺度（interval）、等比尺度（ratio）。其中，名目尺度和順序尺度是定性的、是屬質變數資料，而等距尺度和等比尺度是定量的、適量化變數資料。定量變數，又根據數據是否可數，分為離散的和連續的兩種。

	Variables					
Individuals	Gender (M/F)	Age	Weight (lbs.)	Height (in.)	Smoking (1=No, 2=Yes)	Race
Patient #1	M	59	175	69	1	White
Patient #2	F	67	140	62	2	Black
Patient #3	F	73	155	59	1	Asian
.	.	.	.	.	.	.
.	.	.	.	.	.	.
.	.	.	.	.	.	.
.	.	.	.	.	.	.
.	.	.	.	.	.	.
Patient #75	M	48	90	72	1	White



在R中，依據統計學和電腦科學的習慣，變數類型區分如下：

變數類型	變數類型英文	範例
數值	numeric	2
整數	integer	2L
邏輯值	logical	TRUE
文字	character	"Learning R the easy way"
日期	Date	Sys.Date()
時間	POSIXct POSIXt	Sys.time()

2. R語言的基本運算

2-1. 基本運算-賦值

在深入瞭解不同變數類型之前，我們要先學習一個符號 <- 稱為賦值，它的作用是將符號右邊的值指派給符號左邊的物件。

在 RStudio 中，我們可以按 alt 與 - 幫我們生成 <- 符號。 <- 符號跟其他程式語言的 = 符號功能是完全相同

的。 事實上在 R 語言中使用 = 符號來賦值也是完全沒問題，但是因為絕大多數的 R 語言使用者仍然習慣使用 <- 符號，還是推薦使用 <- 符號。

```
my_num <- 2  #變數賦值
```

在 # 符號後面編寫註解，告訴別人這段程式在做什麼事情

下面將從建立變數、查看變數、判斷變數與轉換變數四方面，依據變數的類型瞭解變數元素的不同特性。

2-2 建立變數類型

2-2-1 變數類型-數值（浮點數）

數值（numeric）包括整數（沒有小數點）與浮點數（有小數點）的數值。 不論我們輸入的數字帶有小數位數或不帶有小數位數，R 語言預設儲存為浮點數。

```
my_num1 <- 2.33
my_num1

my_num2 <- 2.0
my_num2

my_num3 <- 2
my_num3
```

2-2-2 變數類型-整數

當我們輸入一個整數並加入L，R語言就會儲存為整數（integer）。 如果整數帶有不必要的小數位數，R語言會回傳警示訊息，但依舊會儲存為整數；但若在帶有小數位數的數字後加上L，則R 語言回傳警示訊息，並且忽略 L 儲存為數值。

```
my_int1 <- 2L
my_int1

my_int2 <- 2.0L
my_int2

my_int3 <- 2.33L
my_int3
```

2-2-3 變數類型-邏輯值

當我們進行判斷條件或者篩選的時候就會需要使用邏輯值（logical），邏輯值只有 `TRUE` 與 `FALSE` 這兩個值，或者可以簡寫為 `T` 與 `F`。 語言對大小寫是敏感的（case-sensitive），像是 `TRUE` 會被 R 語言識別為邏輯值，但是 `True` 與 `true` 則不會喔！

```
TRUE
FALSE

T
F

True
true
```

除了直接輸入邏輯值，我們也可以透過判斷條件得到邏輯值的輸出：

```
8 > 7 # 判斷 8 是否大於 7
8 < 7 # 判斷 8 是否小於 7
8 >= 7 # 判斷 8 是否大於等於 7
8 <= 7 # 判斷 8 是否小於等於 7
8 == 7 # 判斷 8 是否等於 7
8 != 7 # 判斷 8 是否不等於 7
7 %in% c(8, 7) # 判斷 7 是否包含於一個 c(8, 7) 之中
```

2-2-4 變數類型-文字

在 R 語言中我們可以使用單引號（'）或雙引號（"）來建立文字（character），個人則是習慣使用雙引號（"）來建立文字。 又，若在數字前後加上雙引號，數字也會被儲存為文字形式，無法進行數值的加減乘除等運算。

```
first_name <- "Tony"
first_name
class(first_name)
```

2-2-5 變數類型-日期

在 R 語言中被定義為日期（Date）的變數外觀看起來跟文字沒有什麼差別，但是我們一但將它們放入 `class()` 函數中檢驗，就會發現它並不是文字。

`Sys.Date()` 是一個不需要任何輸入就會輸出電腦系統日期的函數。

```
sys_date <- Sys.Date() # 系統日期
sys_date # 看起來跟文字相同
class(sys_date)
```

2-2-6 變數類型-時間

在 R 語言中被定義為時間（POSIXct POSIXt）的變數外觀看起來跟文字同樣也沒有什麼差別，但是我們一但將它們放入 `class()` 函數中檢驗，就會發現它並不是文字。

`Sys.time()` 是一個不需要任何輸入就會輸出電腦系統時間的函數。

```
sys_time <- Sys.time() # 系統時間
sys_time # 看起來跟文字相同
class(sys_time)
```

隨堂練習題

1. 將您的身高（公分）指派給 my_height；體重（公斤）指派給

```
my_weight
my_height <- ___
my_weight <- ___
```

1. 分別將 my*height 與 my*weight 輸出在命令列

```
___
___
```

1. 利用 my*height 與 my*weight 計算您的身體質量指數（Body Mass Index，BMI），BMI 計算公式為：

```
bmi <- (___) / (___ / 100)^2
bmi
```

1. 將您的姓名指派給 my_name 並且輸出在命令列

```
my_name <- "___"
my_name
```

2-3. 查看變數類型

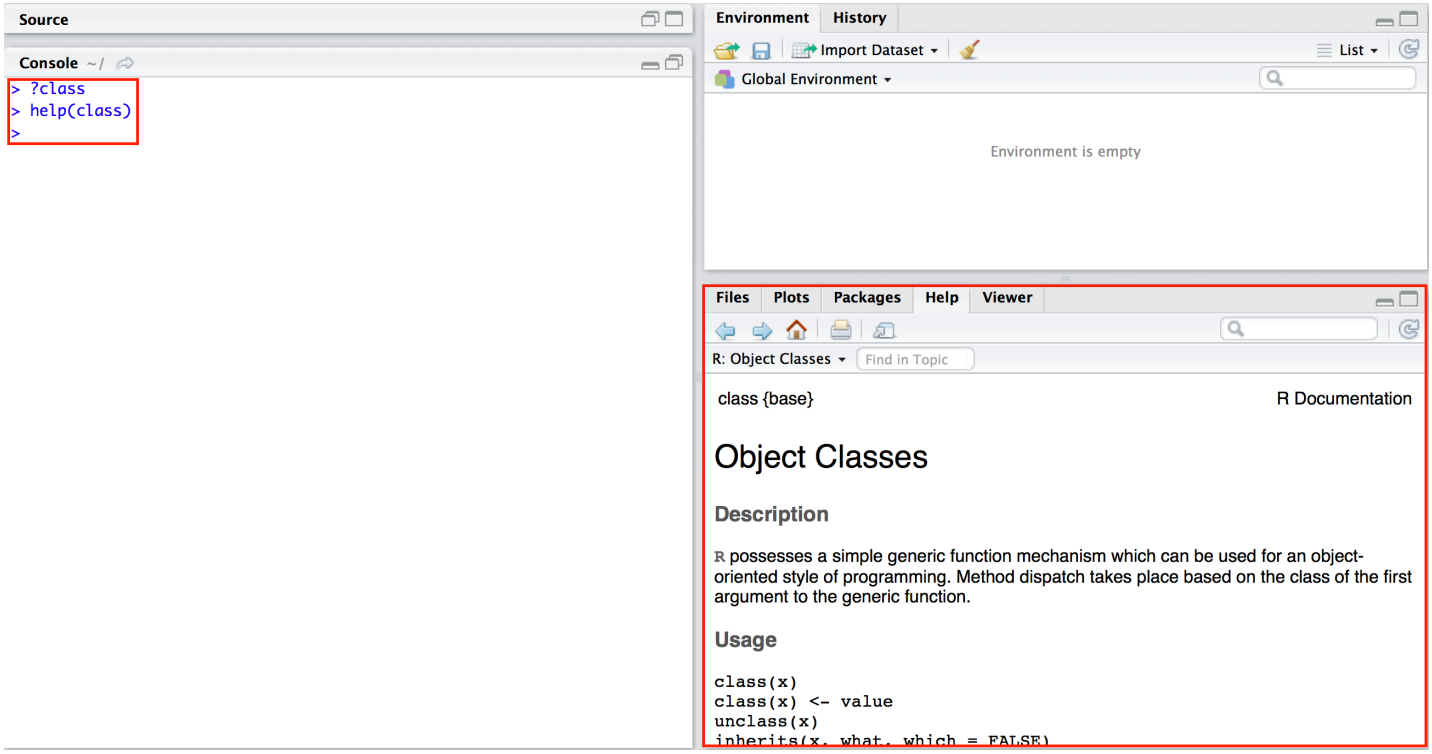
函數 `class()` 可告訴我們輸入的變數是什麼類型。

```
class(2)
class(2L)
class(TRUE)
class("Learning R the easy way")
class(Sys.Date())
class(Sys.time())
```

～前一節中所有的指令都可進行變數類型判斷

```
my_int1 <- 2L
class(my_int1)

my_int2 <- 2.0L
class(my_int2)

my_int3 <- 2.33L
class(my_int3)

class(TRUE)
class(FALSE)

class(T)
class(F)

class(True)
class(true)

first_name <- 'Tony'
first_name
class(first_name)
```

若對任何函數的使用感到好奇，可以在命令列輸入使用 `?` 或 `help()` 進行查詢 。例如， `?class` 或者 `help(class)` ，這兩個指令都會在右下角打開文件。



2-4. 判斷變數類型

R 語言除了使用 `class()` 查詢變數類型外，也使用一系列 `is.類型名稱()` 的函數回傳邏輯值，並用回傳的結果 `TRUE` 或者 `FALSE` 判斷必測類型。

除了判斷日期與時間變數類型要使用的是 `inherits()` 函數，其他變數類型都可以使用 `is.類型名稱()` 這樣形式的函數判斷。

- is.numeric()
- is.integer()
- is.logical()
- is.character()
- inherits(x, what = "Date")
- inherits(x, what = "POSIXct")

2-4-1 判斷是否為數值

使用 `is.numeric()` 函數請 R 語言判斷這個變數是不是數值。

```
is.numeric(8.7)
is.numeric("8.7")
```

2-4-2 判斷是否為整數

使用 `is.integer()` 函數請 R 語言判斷這個變數是不是整數。

```
is.integer(7L)
is.integer(7)
```

2-4-3 判斷是否為邏輯值

使用 `is.logical()` 函數請 R 語言判斷這個變數是不是邏輯值。

```
is.logical(FALSE)
is.logical("FALSE")
```

2-4-4 判斷是否為文字

使用 `is.character()` 函數請 R 語言判斷這個變數是不是文字。

```
is.character("TRUE")
is.character(TRUE)
```

2-4-5 判斷是否為日期

使用 `inherit(x, what = "Date")` 函數請 R 語言判斷這個變數 x 是不是日期，`Sys.Date()` 是一個 Date 的類型，因此 inherit(Sys.Date(), what = "Date") 函數會回傳 TRUE；而 "1970-01-01" 是一個 Character 的類型，inherit("1970-01-01", what = "Date") 函數會回傳 FALSE。

```
inherits(Sys.Date(), what = "Date") # Sys.Date() 是日期類型

inherits("1970-01-01", what = "Date") # "1970-01-01" 是文字類型
```

2-4-6 判斷是否為時間

使用 inherit(x, what = "POSIXct") 函數請 R 語言判斷這個變數 x 是不是時間，Sys.time() 是一個 POSIXct 的類型，因此 inherit(Sys.time(), what = "POSIXct") 函數會回傳 TRUE；而 "1970-01-01 00:00:00" 是一個 Character 的類型，inherit("1970-01-01 00:00:00", what = "POSIXct") 函數會回傳 FALSE。

```
inherits(Sys.time(), what = "POSIXct") # Sys.time() 是時間類型

inherits("1970-01-01 00:00:00", what = "POSIXct") # "1970-01-01 00:00:00" 是文字類型
```

2-5. 轉換變數類型

變數類型的轉換則是透過一系列 `as.類型名稱()` 的函數進行轉換。

- as.numeric()
- as.integer()
- as.logical()
- as.character()
- as.Date()
- as.POSIXct()

2-5-1 轉換成數值（浮點數）

使用 `as.numeric()` 函數轉換變數為數值類型，我們可以輸入整數、邏輯值、日期或時間讓 R 語言轉換成數值。

```
as.numeric(7L)
as.numeric(TRUE)
as.numeric(FALSE)
as.numeric(Sys.Date())
as.numeric(Sys.time())
```

2-5-2 轉換成整數

使用 `as.integer()` 函數轉換變數為數值類型，我們可以輸入沒有小數位數的數值、邏輯值、日期或時間讓 R 語言轉換成整數。

```
as.integer(7)
as.integer(TRUE)
as.integer(FALSE)
as.integer(Sys.Date())
as.integer(Sys.time())
```

2-5-3 轉換成邏輯值

使用 `as.logical()` 函數轉換變數為邏輯值類型，輸入數值或整數類型的 0 會轉換成為 FALSE，其他的數字則一律轉換為 TRUE。

輸入文字類型的 "TRUE"、"True" 或 "true" 則會轉換成為 TRUE，反之亦同。

```
as.logical(0)
as.logical(0L)
as.logical(1L)
as.logical(-1.3)
as.logical(87)

as.logical("TRUE")
as.logical("True")
as.logical("true")
as.logical("FALSE")
as.logical("False")
as.logical("false")
```

2-5-4 轉換成文字

使用 `as.character()` 函數轉換變數為文字類型，我們可以輸入任意的變數類型讓 R 語言轉換成文字。

```
as.character(8.7)
as.character(87L)
as.character(TRUE)
as.character(Sys.Date())
as.character(Sys.time())
```

2-5-5 轉換成日期

使用 `as.Date()` 函數可以轉換文字變數為日期類型，而 as.Date() 函數預設可以識別的格式為 %Y-%m-%d 或 %Y/%m/%d 這兩種。

如果是其他非預設的文字變數格式，必須要加入 format 參數告知日期被記錄的文字變數格式為何，函數才能順利轉換，例如將月與日的資訊寫在年份的前面格式，若沒有以 format 參數指定就會轉換成錯誤的日期：

```
as.Date("1970-01-01")
as.Date("1970/01/01")

as.Date("01-01-1970") # 轉換錯誤
as.Date("01-01-1970", format = "%m-%d-%Y") # 轉換正確
as.Date("01/01/70") # 不是預設格式，轉換失敗
as.Date("01/01/70", format = "%m/%d/%y") # 轉換正確
```

符號 格式

- `%d` 日
- `%a` 禮拜幾的縮寫

- `%A` 禮拜幾
- `%m` 月
- `%b` 月名稱的縮寫
- `%B` 月名稱
- `%y` 兩位數的年
- `%Y` 四位數的年

2-5-6 轉換成時間

使用 `as.POSIXct()` 函數可以轉換文字變數為時間類型，如果沒有指定參數，tz = 會預設使用電腦的時區。

```
as.POSIXct("1970-01-01 00:00:00")
as.POSIXct("1970-01-01 00:00:00", tz = "GMT")
```

2-6 日期與時間的變數轉換與運算(Optional)

日期變數類型是可以被轉換為整數，而文字是不行的。 日期變數類型是可以進行四則運算，而文字類型不行。

- 如果我們使用as.character() 與 as.integer() 函數來進行變數類型轉換，文字類型的系統日期與時間在轉換整數時會失敗，而產生一個遺失值（Not Available，NA）。

Question: 日期變數類型的系統日期如何轉換為整數？時間變數類型的系統時間如何轉換為整數？有什麼準則嗎？

R 語言預設以西元 1970 年 1 月 1 日作為 0，在這一天以後的每天都 +1 來記錄，而這一天以前的每天都 -1 來記錄。

```
sys_date <- Sys.Date() # 系統日期
sys_date # 看起來跟文字相同
class(sys_date)

sys_date <- Sys.Date()
sys_date_char <- as.character(sys_date) # 創造一個文字類型

as.integer(sys_date)
as.integer(sys_date_char)

date_of_origin <- as.Date("1970-01-01")
as.integer(date_of_origin)
as.integer(date_of_origin + 1)
as.integer(date_of_origin - 1)

date_of_origin
date_of_origin + 1
date_of_origin - 1

sys_date <- Sys.Date()
sys_date_char <- as.character(sys_date) # 創造一個文字類型

sys_date - 1 # 昨天的日期
sys_date_char - 1
```

R 語言預設以西元 1970 年 1 月 1 日格林威治標準時間（Greenwich Mean Time，GMT）00 時 00 分 00 秒作為 0，在這個時間點以後的每秒都 +1 來記錄，這個時間點以前的每秒都 -1 來記錄。

我們這裡所使用的參數 tz = "GMT" 是指定時區，假如您的電腦和我一樣時區是設在中原標準時間（Chungyuan Standard Time，CST），早格林威治標準時間八個小時（GMT + 8），則基準時間會是西元 1970 年 1 月 1 日 08 時 00 分 00 秒。

```
sys_time <- Sys.time() # 系統時間
sys_time # 看起來跟文字相同
class(sys_time)

sys_time <- Sys.time()
as.integer(sys_time)

time_of_origin <- as.POSIXct("1970-01-01 00:00:00", tz = "GMT")
as.integer(time_of_origin)
as.integer(time_of_origin + 1)
as.integer(time_of_origin - 1)

time_of_origin
time_of_origin + 1
time_of_origin - 1

time_of_origin_cst <- as.POSIXct("1970-01-01 08:00:00")
as.integer(time_of_origin_cst)
```

～日期與字串的相關轉換操作，可以考慮使用簡單易懂的lubridate套件。如果想要將年/月/日格式的文字轉換為日期物件，可使用ymd()函數（y表年year，m表月month，d表日day）；如果想要將月/日/年格式的文字轉換為日期物件，則使用mdy()函數，以此類推。

隨堂練習題

1. 香港搖滾樂隊 Beyond 於 1983 年成立，我們假設成立日期是 1983-12-31，請將成立日期指派給 beyond_start 並轉換成整數輸出在命令列

```
beyond_start <- as.Date("___")
as.integer(___)
```

1. 請以系統日期計算今年是 Beyond 成立幾週年紀念？

```
beyond_start <- as.Date("___")
days_diff <- ___ - ___ # 計算天數差距
years_diff <- ___ / 365 # 除以 365 換算成年
```

1. 1999 年 9 月 21 日 1 時 47 分 16 秒發生震央位於南投縣集集鎮，芮氏規模 7.3 的地震，請以文字記錄這個時間，並將它指派給 major*quake*time

```
major_quake_time <- "___"
```

1. 1999 年 9 月 21 日 1 時 57 分 15 秒發生第一個芮氏規模超過 6 的餘震，請以文字記錄這個時間，並將

它指派給 first*aftershock*time

```
first_aftershock_time <- "___"
```

1. 請將前兩題的生成的變數轉換為時間類型，計算間隔多久發生第一個芮氏規模超過 6 的餘震

```
major_quake_time <- as.POSIXct(___)
first_aftershock_time <- as.POSIXct(___)
___ - ___
```

3. 基本運算

運算子名稱	符號	說明	備註
算術運算子	+	加法	
	-	減法	
	*	乘法	
	**	次方	
	/	除法	盡量少用，因爲耗費資源很大
	%	取餘數	盡量少用，因爲耗費資源很大
二元運算子	~	NOT	
	&	AND	
	|	OR	
	^	XOR	
邏輯運算子	!	邏輯 NOT	回傳 True / False
	&&	邏輯 AND	回傳 True / False
	||	邏輯 OR	回傳 True / False
比較運算子	>	大於	">="邏輯電路比">"簡單
	>=	大於等於	
	<	小於	"<="邏輯電路比"<"簡單
	<=	小於等於	
	==	等於	
	!=	不等於	
位移運算子	<<	左移	相當於乘 2^n，n 爲位移數
	>>	右移	相當於除 2^n，n 爲位移數
條件運算子	?:	if-else	等同於 if-else
連結運算子	{}	連接變數	

3-1. 數學運算子

對數值，整數與邏輯值進行數學運算，常用的數學運算子有六種：

- `+` 加
- `-` 減
- `*` 乘
- `/` 除

- `^` 或 `**` 次方
- `%%` 回傳餘數

為什麼 `TRUE` 和 `FALSE` 納入四則運算沒有任何問題呢？因為在 R 語言中，TRUE 跟 1 或者 1L 是相等的；FALSE 跟 0 或者 0L 是相等的。

```
# 數值的運算

first_num <- 8
second_num <- 7

first_num + second_num
first_num - second_num
first_num * second_num
first_num / second_num

first_num^second_num
first_num %% second_num

# 整數的運算

first_int <- 8L
second_int <- 7L

ans <- first_int * second_int
ans
class(ans)

ans <- first_int / second_int
ans
class(ans)

# 數值與整數的運算

my_num <- 8
my_int <- 7L

ans <- my_num + my_int
class(ans)
```

3-2. 邏輯運算子

常用之邏輯判斷也可在R中直接使用

- 大於 `>`
- 小於 `<`

- 等於 `==` ，為了不與變數設定混淆，判斷兩變數是否相等，要用雙等號
- 大於等於 `>=`
- 小於等於 `<=`

```
# 邏輯值的運算

my_num <- 0
my_int <- 0L
my_logi <- FALSE
my_num == my_logi
my_int == my_logi

my_num <- 1
my_int <- 1L
my_logi <- TRUE
my_num == my_logi
my_int == my_logi

# 數值、整數與邏輯值的運算

my_logi <- TRUE
ans <- my_num + my_int + my_logi
ans
class(ans)

my_logi <- FALSE
ans <- my_num + my_int + my_logi
ans
class(ans)
```

3-3. 其他

邏輯混合判斷

在R中使用單符號即可表示:

- 且 `&`
- 或 `|`
- 反向布林變數 `!`

錯誤訊息

- Message：有可能的錯誤通知，程式會繼續執行
- Warning：有錯誤，但是不會影響太多，程式會繼續執行
- Error：有錯，而且無法繼續執行程式

- Condition：可能會發生的情況

R 語言的資料結構

4. 資料結構分類

- 一維
  - 向量（vector）
  - 因素（factor）
- 二維
  - 矩陣（matrix）
  - 資料框架（data.frame）
- 多維
  - 陣列（array）
  - 清單（list）

4-1. 一維資料結構：向量vector

向量是由一種變數類型組成。

建立向量

- 使用 `c()` 函數將我們想要放入的變數元素資料集結在一個向量之中。
- 使用 `rep()` 函數可以生成重複變數的向量， `times` 參數可以指定要生成幾個。
- 使用 `seq()` 函數可以生成等差數列， `from` 參數指定數列的起始值， `to` 參數指定數列的終止值， `by` 參數指定數值的間距。
- 使用 `:` 快速生成數值間距為 1 的數列。

```
# 使用C函數集結元素

x <- c(1,2,3,4)  #數字向量
x

season_1 <- "spring"
season_2 <- "summer"
season_3 <- "autumn"
season_4 <- "winter"

four_seasons <- c("spring", "summer", "autumn", "winter") #文字向量
four_seasons


rep("2", times = 10)
rep("R", times = 10)

seq(from = 1, to = 10, by = 1) #等差函數

1:10
```

向量有一個很特別的屬性，那就是只能包含一種變數類型。

- 如果我們將整數跟數值放入同一個向量之中，那麼向量會將整數轉換成浮點數。
- 如果我們將邏輯值跟整數放入同一個向量之中，那麼向量會將邏輯值轉換成整數。
- 如果我們將邏輯值跟數值放入同一個向量之中，那麼向量會將邏輯值轉換成浮點數。
- 如果我們將邏輯值、整數、數值放入同一個向量之中，那麼向量會全部轉換成數值。
- 如果我們將邏輯值、整數、數值還有文字放入同一個向量之中，那麼向量會全部轉換成文字。

```
lucky_numbers <- c(7L, 24)
class(lucky_numbers[1])

lucky_numbers <- c(7L, FALSE)
lucky_numbers
class(lucky_numbers[2])

mixed_vars <- c(TRUE, 7L, 24, "spring")
mixed_vars
class(mixed_vars[1])
class(mixed_vars[2])
class(mixed_vars[3])
```

從向量變數選出元素變數

- 直接使用中括號搭配索引值從向量選出元素。
- 用判斷運算子來對向量進行篩選。
    - 如果不只一個條件，我們可以使用 `&（AND）` 以及 `|（OR）` 這兩個符號連結判斷條件。

```
#方法一：中括號搭配索引值

favorite_season <- four_seasons[3]
favorite_season

favorite_seasons <- four_seasons[c(-2, -4)] # 我喜歡春天或秋天
favorite_seasons

#方法二：判斷運算子

my_favorite_season <- four_seasons == "autumn"
four_seasons[my_favorite_season]

my_favorite_seasons <- four_seasons == "spring" | four_seasons == "autumn" # 我喜歡春
four_seasons[my_favorite_seasons]
```

向量運算

向量可直接做加減乘除運算，向量和向量之間也可做運算。

```
numvec<-1:10 # c(1,2,3,4,5,6,7,8,9,10)

numvec+3 # 所有元素+3
numvec*2 # 所有元素*2

numvec1<-1:3 ## c(1,2,3)
numvec2<-4:6 ## c(4,5,6)
numvec1+numvec2
numvec1*numvec2
```

4-2. 一維資料結構：因素factor

因素是由文字向量轉換而成，用於表示類別變數數據。可使用factor(資料向量)進行宣告。

通常，因素向量需要載明分類的類別，並說明此類別分類是名目尺度還是順序尺度。

- 我們可以使用levels=類別次序，用levels參數設定分類類別。
    - 類別的種類與數目一但決定，通常不會再作更動。
- 我們也可以使用 `ordered = TRUE` 或 `ordered = FALSE` 來表示順序尺度或名目尺度。

- 若沒有寫此參數， R 語言會預設使用字母順序排序，此時可能會產生不符合直覺的排序。

比如大學中有大學生、碩士班學生與博士班學生三種類別的學生，使用方法為factor(資料向量,levels=類別次序)，levels參數可設定各類別的次序。任和元素都要是大學生、碩士班學生與博士班學生其中一種。

```
factor(c("大學生","碩士班學生","博士班學生"),
       levels = c("大學生","碩士班學生","博士班學生"))
       ordered = FALSE
 ```

～因素是一個帶有層級（Levels）資訊的向量，我們使用 `factor()` 函數可以將向量轉換成因素向量，輸出因


```{r}
four_seasons <- c("spring", "summer", "autumn", "winter")
four_seasons

four_seasons_factor < factor(four_seasons)
four_seasons_factor

four_seasons_factor <- factor(four_seasons, ordered = TRUE, levels = c("summer", "wi
four_seasons_factor

temperatures <- c("warm", "hot", "cold")
temp_factors <- factor(temperatures, ordered = TRUE, levels = c("cold", "warm", "hot
temp_factors

temperatures <- c("warm", "hot", "cold")
temp_factors <- factor(temperatures, ordered = TRUE)
temp_factors
```

隨堂練習題

1. 我們有一個文字向量 weekdays 是一週的星期一到星期五，請您將最喜歡的週五（Happy Friday）從這個向量中用索引值選出來並且指派給 favorite_day。

```
weekdays <- c("Monday", "Tuesday", "Wednesday", "Thursday", "Friday")
favorite_day <- weekdays[___]
favorite_day
```

1. 同樣的文字向量，請您利用判斷運算子將最藍的週一（Blue Monday）從這個向量中剔除後將剩餘的日子指派給 without_monday。

```
weekdays <- c("Monday", "Tuesday", "Wednesday", "Thursday", "Friday")

not_blue_monday <- weekdays != "Monday"
without_monday <- weekdays[___]
without_monday
```

1. 我們有一個文字向量 speed*char 描述速度的快慢，請您使用 factor() 函數轉換成因素向量 speed*factor，並且讓快慢有排序（慢 < 快）

```
speed_char <- c("slow", "fast")
speed_factor <- ___(speed_char, ordered = ___, levels = c("___", "___"))
speed_factor
```

4-3 二維資料結構:矩陣matrix

矩陣是能夠儲存列（Row）與欄（Column）的資料結構。Row 指的是水平方向資料，Column 指的是垂直方向資料。習慣是先 Row 後 Column。

一個矩陣的大小以 mxn表示，這個矩陣的外觀具有 m 個水平方向資料，n 個垂直方向資料。

建立矩陣

- 使用 `matrix()` 函數，可指定參數 `nrow` `ncol` 表示列數與行數，也可使用 `byrow` 的參數指定要用什麼順序擺放原先在向量中的元素。 `byrow` 參數的預設值為 `FALSE` ，如果我們沒有特別指定，就是以垂直的方向來擺放矩陣。

```
my_mat <- matrix(1:6, nrow = 2)
my_mat

my_mat <- matrix(1:6, nrow = 2, ncol = 3, byrow = TRUE)
my_mat
```

矩陣與向量有一點很相似的特性，那就是包含一種變數類型：

- 如果我們將整數跟數值放入同一個矩陣之中，那麼矩陣會將整數轉換成浮點數。
- 如果我們將邏輯值跟整數放入同一個矩陣之中，那麼矩陣會將邏輯值轉換成整數。
- 如果我們將邏輯值跟數值放入同一個矩陣之中，那麼矩陣會將邏輯值轉換成浮點數。
- 如果我們將邏輯值、整數、數值放入同一個矩陣之中，那麼矩陣會全部轉換成數值。
- 如果我們將邏輯值、整數、數值還有文字放入同一個矩陣之中，那麼矩陣會全部轉換成文字。

```
my_mat <- matrix(c(1, 2, TRUE, FALSE, 3, 4), nrow = 2)
my_mat

class(my_mat[, 2]) # 原本第二欄（2nd column）的輸入是兩個邏輯值
```

從矩陣選出變數

- 矩陣同樣能夠以 `[ ]` 中括號搭配索引值選出裡面的變數，跟向量不同的是，現在有兩個維度的索引值必須指定，所以要用 `[m, n]` 兩個索引值來搭配選擇。
- 可以透過判斷運算子來對矩陣進行篩選。

```
my_mat <- matrix(1:6, nrow = 2)
my_mat

#方法一：中括號搭配索引值

my_mat[2, 3] # 選出位於（2, 3）這個位置的 6
my_mat[2, ] # 選出所有第二列（2nd row）的元素
my_mat[, 3] # 選出所有第三欄（2nd column）的元素

#方法二：判斷運算子

filter <- my_mat < 6 & my_mat > 1
my_mat[filter]
```

4-4 二維資料結構:資料框架data frame

資料框絕能夠容許不同的欄位有不同的變數類型，R讀取外部資料之後，通常預設以資料框的格式儲存。常見的Excel試算表就是類似的資料表現形式。我們習慣使用觀測值（Observations，Obs）來稱呼資料框中水平方向的資料，使用變數（Variables）來稱呼資料框中垂直方向的資料。

建立資料框架

- 運用 data.frame()函數手動創造資料框

例：讓我們建立一個很簡單的資料框叫做 `great_nba_teams` ，這個資料框有隊名、勝場數、敗場數、是否獲得總冠軍與球季。

- 除了將資料框直接輸出在命令列（Console），我們可以使用 View()函數瀏覽資料框的外觀與內容。

```
team_name <- c("Chicago Bulls", "Golden State Warriors")
wins <- c(72, 73)
losses <- c(10, 9)
is_champion <- c(TRUE, FALSE)
season <- c("1995-96", "2015-16")

great_nba_teams <- data.frame(team_name, wins, losses, is_champion, season)
View(great_nba_teams)
```

資料框不像矩陣僅能包含一種變數類型

在我們的例子 `great_nba_teams` 資料框中，有文字（`team_nam`e、season）、整數（wins、losses）與邏輯值（`is_champion`），我們可以用 `str()` 函數觀察資料框的變數類型。

```
great_nba_teams <- data.frame(team_name, wins, losses, is_champion, season, stringsA
str(great_nba_teams)
```

資料框預設會將文字的內容以因素向量儲存。

如果您希望將因素向量修正為文字，可以有下面兩種作法:

- 一種是建立的時候在 `data.frame()` 函數指定 `stringsAsFactors = FALSE` 。
- 另一種是事後使用 `as.character()` 函數進行變數類型的轉換。

```
team_name <- c("Chicago Bulls", "Golden State Warriors")
wins <- c(72, 73)
losses <- c(10, 9)
is_champion <- c(TRUE, FALSE)
season <- c("1995-96", "2015-16")

#方法一：stringsAsFactors = FALSE
great_nba_teams <- data.frame(team_name, wins, losses, is_champion, season, stringsA

#方法二:as.character()
great_nba_teams <- data.frame(team_name, wins, losses, is_champion, season)
great_nba_teams[, 1] <- as.character(great_nba_teams[, 1])
great_nba_teams[, 5] <- as.character(great_nba_teams[, 5])
```

從資料框架選出變數

- 資料框架同樣能夠以 `[m, n]` 中括號搭配兩個索引值來選擇出變數。
- 資料框架支援使用變數名稱來選擇，可用 `[, "變數名稱"]` 寫法。
- 資料框架支援使用變數名稱來選擇，可用 `$變數名稱` 寫法。
- 可以透過判斷運算子來對資料框架進行篩選。

～資料框做欄位資料擷取後，可當成向量做資料編輯。

```
team_name <- c("Chicago Bulls", "Golden State Warriors")
wins <- c(72, 73)
losses <- c(10, 9)
is_champion <- c(TRUE, FALSE)
season <- c("1995-96", "2015-16")

great_nba_teams <- data.frame(team_name, wins, losses, is_champion, season)

#方法一：中括號搭配索引值
great_nba_teams[1, 1] # 選出第一個變數的第一個觀測值 "Chicago Bulls"
great_nba_teams[1, ] # 選出第一個觀測值
great_nba_teams[, 1] # 選出第一個變數

#方法二：使用變數名稱來選擇
great_nba_teams[, "season"]

# **方法三：使用變數名稱與$來選擇**
great_nba_teams$team_name

#方法四：使用判斷運算子
great_nba_teams <- data.frame(team_name, wins, losses, is_champion, season, stringsA
filter <- great_nba_teams$is_champion == TRUE
great_nba_teams[filter, ] # 注意這個逗號
```

補充：data.table (Optional)

data.table是data.frame資料框資料類型別的延伸，要使用須安裝data.table套件 (Dowle and Srinivasan 2016) 。使用data.table讀取大型資料的速度比使用資料框快上數倍，進階處理語言也相當好用。

隨堂練習題

1. 我們有一個矩陣叫做 my_mat，它是一個 3x3 的矩陣，裡面有 1 到 9 這些數字，請您利用 [m, n] 把 8 選出來。

```
my_mat <- matrix(1:9, nrow = 3)
my_mat[___, ___]
```

1. 同樣的一個矩陣，請您利用判斷運算子來對矩陣進行篩選，選出奇數（1、3、5、7、9）。

```
my_mat <- matrix(1:9, nrow = 3)
filter <- my_mat %% 2 == ___
my_mat[filter]
```

1. 我們繼續使用 `great_nba_teams` 這個資料框，請您分別利用 $變數名稱 與 [, "變數名稱"] 將 `is_champion` 變數挑出來。

```
team_name <- c("Chicago Bulls", "Golden State Warriors")
wins <- c(72, 73)
losses <- c(10, 9)

is_champion <- c(TRUE, FALSE)
season <- c("1995-96", "2015-16")

great_nba_teams <- data.frame(team_name, wins, losses, is_champion, season, stringsA

# 利用 `$變數名稱`
great_nba_teams$___

# 利用`[, "變數名稱"]`
greate_nba_teams[, "___"]
```

4-5. 多維資料結構:陣列array

陣列（array）是矩陣的強化版，除了原有的水平方向資料（Row）與垂直方向資料（Column，）我們可以再多指定一個維度。簡單來說，就是在一個陣列的物件之中，可以允許我們儲存多個矩陣。

4-5-1. 建立陣列

```
my_arr <- array(1:20, dim = c(2, 2, 5))
my_arr
```

從陣列選出變數

- 使用中括號搭配索引值選擇時，需要有三個維度的索引值，分別為第幾列、第幾行與第幾個矩陣。如此可以選擇出矩陣、變數、觀察值與元素等。

```
my_arr <- array(1:20, dim = c(2, 2, 5))

my_arr[, , 1] # 第一個矩陣
my_arr[, , 2] # 第二個矩陣
my_arr[, , 3] # 第三個矩陣
my_arr[, , 4] # 第四個矩陣
my_arr[, , 5] # 第五個矩陣

my_arr[1, , 2] # 選出第二個矩陣的第一個 row（觀察值）
my_arr[, 2, 2] # 選出第二個矩陣的第二個 column （變數）

my_arr[1, 2, 2] # 選出 7（元素）
```

4-6. 多維資料結構:清單list

清單（list）是一個終極的巨大資料型容器，可以收納所有的R語言物件。 它除了可包括各種元素如數值與文字外，也可以包含一維資料結構、二維資料結構與多維資料結構等物件。

建立清單

- 利用 `list()` 函數收納不同類型的物件。

```
# 單變數元素如文字數字、一維文字數字向量
listSample<-list(Students=c("Tom","Kobe","Emma","Amy"),Year=2017,
                 Score=c(60,50,80,40),School="CGU")
listSample
```

```
# 例：單變數元素、一維數字與文字向量、二維矩陣與資料框
title <- "Great NBA Teams"
teams <- c("Chicago Bulls", "Golden State Warriors")
wins <- c(72, 73)
losses <- c(10, 9)
winning_percentage <- wins / (wins + losses)
season <- c("1995-96", "2015-16")

wins_losses <- matrix(c(wins, losses), nrow = 2)

df <- data.frame(Teams = teams, Winning_Percentage = winning_percentage, Season = se

great_nba_teams <- list(title, teams, wins_losses, df)
great_nba_teams
```

從清單選出變數

- 要選擇清單裡面的物件，要用 `[[]]` 雙中括號選擇出物件。
- 建立清單的同時，如果指定了物件的命名，就可以在選擇時使用 `$物件名稱` 。

```
list(1,2,3)
[[1]]
[[2]]
[[3]]

list(c(1,2,3))
[[1]]

list3 <- list(c(1,2,3), 3:7)
[[1]]
[[2]]
```

```
#方法一：用雙中括號選

great_nba_teams[[1]] # 選出清單中的第一個物件
great_nba_teams[[2]] # 選出清單中的第二個物件
great_nba_teams[[3]] # 選出清單中的第三個物件
great_nba_teams[[4]] # 選出清單中的第四個物件

great_nba_teams[[3]][1, ] # 選出芝加哥公牛隊 1995-96 的戰績
great_nba_teams[3][1, ] # 選出芝加哥公牛隊 1995-96 的戰績會產生錯誤

#方法二：用`$物件名稱`# 有點問題
great_nba_teams$teams # 選出 teams 這個向量
great_nba_teams$df # 選出 df 這個資料框

great_nba_teams$teams[1, ] # 選出 teams 這個向量選出芝加哥公牛隊 1995-96 的戰績
great_nba_teams[[4]]$winning_percentage # 選出創紀錄的球季勝率
```

R 語言有很多非常便利的函數，這些函數的輸出多半是一個清單。

假如我們有兩個向量 x 與 y，兩個向量的關係是 $y = 2x + 5$。

- 我們可以利用 lm() 這個函數建立一個 lm_fit 清單。
- 從這個清單中，我們可以取出裡面的coefficients這個物件（是一個向量），就可以知道 x 跟 y 的關係，跟我們建立時的設定完全相同。

```
x <- 1:10
y <- 2 * x + 5

lm_fit <- lm(formula = y ~ x)
lm_fit$coefficients
lm_fit$coefficients[1] # 截距為 5
lm_fit$coefficients[2] # x 係數為 2
```

隨堂練習題

1. 1. 我們把 1 到 1000 儲存在 10 個 10x10 的矩陣，並且收納在一個陣列 my_arr 之中，請你練習用索引值將 315 這個數字選出來

```
my_arr <- array(1:1000, dim = c(10, 10, 10))
my_arr[___, ___, ___]
```

1. 我們已經建立好幾個物件，請您幫我們收納進一個清單中叫做 worst*nba*teams，並利用 [[索引值]] 從清單中選出 wins 這個向量

```
title <- "Worst NBA Teams"
teams <- c("Charlotte Bobcats", "Philadelphia 76ers")
wins <- c(7, 9)
losses <- c(59, 73)
worst_nba_teams <- list(___, ___, ___, ___)
worst_nba_teams[[___]]
```

1. 同樣的幾個物件，請您建立清單 worst*nba*teams 的時候為每個物件命名，並且利用 $物件名稱 從清單中選出 teams 這個向量

```
title <- "Worst NBA Teams"
teams <- c("Charlotte Bobcats", "Philadelphia 76ers")
wins <- c(7, 9)
losses <- c(59, 73)

worst_nba_teams <- list(___ = title, ___ = teams, ___ = wins, ___ = losses)

worst_nba_teams$___
```

Summary:資料屬性查詢函數

資料屬性可透過下列函數查詢:

- 名稱：透過names()函數，可取得各種資料之名稱。

- 各維度名稱：透過dimnames()函數，可顯示資料框列與行的名稱，先顯示列、再顯示行。
- 長度：透過length()函數，可顯示資料長度。
- 各維度長度：透過dim()函數，可顯示資料框列與行的長度，先顯示列、後顯示行。
- 資料型態：使用class()函數，可知道變數類別。
- 各類資料計數：使用table()函數，可知道向量中每個值出現幾次。
- 總覽資料：使用str()函數，可總覽變數資訊。

5. 線性代數

5-1. 創建矩陣

matrix() 函數

使用 `matrix(data, nrow, ncol)` 函數創建矩陣，data 參數可以輸入一個值或者一組值， `nrow` 參數輸入矩陣的列數， `ncol` 參數輸入矩陣的欄數；如此一來就能將矩陣的外觀指定妥當，我們習慣稱一個具有 m 列 n 欄的矩陣其外觀為 `m x n` 。

```
# 建立一個 3 x 4 的矩陣，其中的數字皆是 24
my_mat <- matrix(24, nrow = 3, ncol = 4)
my_mat

# 建立一個 2 x 3的矩陣，其中的文字皆是 Luke Skywalker
luke_mat <- matrix("Luke Skywalker", nrow = 2, ncol = 3)
luke_mat

# 建立一個 4 x 2 的矩陣，其中的邏輯值皆是 TRUE
true_mat <- matrix(TRUE, nrow = 4, ncol = 2)
true_mat
```

data 參數除了數字，也可以輸入文字或者邏輯值。例如建立一個 2 x 3的矩陣，其中的文字皆是 Luke Skywalker。 或者建立一個 4 x 2 的矩陣，其中的邏輯值皆是 TRUE。

c() 函數

相較在 data 參數輸入一個值，更常見的是透過 `c()` 函數輸入一組值。例如將 5, 3, 2, 17 這四個數字放入一個 2 x 2 的矩陣中。

```
# 將 5, 3, 2, 17 這四個數字放入一個 2 x 2 的矩陣
conf_mat <- matrix(c(5, 3, 2, 17), nrow = 2, ncol = 2)
conf_mat

# 將 5, 3, 2, 17 這四個數字放入一個 2 x 2 的矩陣，指定 byrow 參數為 TRUE
conf_mat <- matrix(c(5, 3, 2, 17), nrow = 2, ncol = 2, byrow = TRUE)
conf_mat
```

預設擺放數字進入矩陣的順序是先填完第一欄、再填入第二欄；如果希望將順序調整為先填完第一列、再填入第二列，可以指定 byrow 參數為 TRUE。

5-2. 拆解矩陣

利用 `[m, n]` 位置

在 R 語言裡頭，中括號 `[]` 最重要的作用是從物件中選取元，素為我們實踐化整為零的操作。由於矩陣具有兩個維度（列與欄），在中括號 `[]` 之中要放入這兩個維度的索引值，並以逗號分隔，前面放置列的索引值、後面放置欄的索引值。

```
# 分別選出矩陣中的數字
conf_mat <- matrix(c(5, 3, 2, 17), nrow = 2, ncol = 2)
conf_mat[1, 1]
conf_mat[2, 1]
conf_mat[1, 2]
conf_mat[2, 2]

# 選擇一整列或一整欄
conf_mat <- matrix(c(5, 3, 2, 17), nrow = 2, ncol = 2)
conf_mat[1,]
conf_mat[2,]
conf_mat[,1]
conf_mat[,2]
```

如果想要選擇一整列或一整欄，參考矩陣左方與上方的標示，在中括號 `[]` 僅放入列或欄的索引值即可。

利用列標籤、欄標籤

矩陣預設左方與上方的標示，可以讓我們透過兩種方式替代成列標籤與欄標籤。

第一種方式是在 `matrix()` 函數中指定 `dimname` 參數。例如在列標籤標註 Predicted Positive、Predicted Negative，在欄標籤標註 Condition Positive、Condition Negative。

```
# 在列標籤標註 Predicted Positive、Predicted Negative，在欄標籤標註 Condition Positive、C

conf_mat <- matrix(c(5, 3, 2, 17), nrow = 2, ncol = 2, dimnames = list(c("Predicted
conf_mat
```

第二種方式是對已經創建好的矩陣應用 `rownames()` 與 `colnames()` 函數，分別加入列標籤與欄標籤。

一但加入列標籤與欄標籤之後，除了可以繼續沿用 `[m, n]` 拆解矩陣，也能夠應用 [列標籤, 欄標籤] 拆解矩陣。

```
# 對已經創建好的矩陣應用 rownames() 與 colnames() 函數
conf_mat <- matrix(c(5, 3, 2, 17), nrow = 2, ncol = 2)
rownames(conf_mat) <- c("Predicted Positive", "Predicted Negative")
colnames(conf_mat) <- c("Condition Positive", "Condition Negative")
conf_mat

# 應用 [列標籤, 欄標籤] 拆解矩陣
conf_mat <- matrix(c(5, 3, 2, 17), nrow = 2, ncol = 2, dimnames = list(c("Predicted
conf_mat["Predicted Positive", "Condition Positive"] # 5
conf_mat[1, 1] # 5
conf_mat["Predicted Negative", "Condition Negative"] # 5
conf_mat[2, 2] # 17
```

利用條件判斷

當我們面對較龐大的矩陣時，透過位置或列欄標籤來拆解就顯得較緩慢，這時往往會透過撰寫條件判斷來產生邏輯值（logical）放置到中括號裡頭，篩選出判斷為 TRUE 的值。例如將 5 x 5 的矩陣中的偶數篩選出來。

```
# 將 5 x 5 的矩陣中的偶數篩選出來
my_mat <- matrix(1:25, nrow = 5, ncol = 5)
is_even <- my_mat %% 2 == 0
my_mat[is_even]
```

5-3. 特殊矩陣

單位矩陣（Identity Matrix）

對角線上為 1 其餘位置為 0 的矩陣，透過 `diag(nrow, ncol)` 函數建立出單位矩陣。

```
# 透過 diag(nrow, ncol) 函數建立出單位矩陣
diag(nrow = 2, ncol = 2) # 2 x 2 的單位矩陣
diag(nrow = 3, ncol = 3) # 3 x 3 的單位矩陣
diag(nrow = 4, ncol = 4) # 4 x 4 的單位矩陣
```

轉置矩陣（Transpose）

將矩陣 A 中透過 `t()` 函數建立轉置矩陣，每個數字從 (m, n) 的位置轉換到 (n, m) 後，所呈現的矩陣 B 稱為矩陣 A 的轉置矩陣。

```
# 透過 t() 函數建立轉置矩陣
A <- matrix(11:16, nrow = 2, ncol = 3)
B <- t(A)
A
B
```

反矩陣（Inverse)

如果矩陣 M 是一個可逆矩陣（invertible），則矩陣 M 與其反矩陣相乘之後可以得到一個單位矩陣，透過 `solve()` 函數可以取得對稱矩陣的反矩陣。

```
# 透過 solve() 函數可以取得對稱矩陣的反矩陣
M <- matrix(c(4, 2, -7, -3), nrow = 2, ncol = 2)
M_inv <- solve(M)
M_inv


# 並不是每個矩陣都具有反矩陣
M <- matrix(c(8, 12, 2, 3), nrow = 2, ncol = 2)
tryCatch(solve(M), error = function(e){
  print("矩陣為不可逆矩陣")
})
```

- 並不是每個矩陣都具有反矩陣，當矩陣 M 為不可逆（singular）矩陣，會得到一個錯誤。

5-4. 矩陣的運算

`*` 符號用來做矩陣元素級別的相乘，如果要計算內積我們需要使用 `%*%` 符號；並非任意兩個矩陣都能夠相乘，如果矩陣 A 的大小為 m x n、矩陣 B 的大小為 p x q，n 與 p 要相等， A * B 才能夠運算，A * B 的大小為 m x q。

矩陣與矩陣相乘（內積）

```
# 計算內積我們需要使用 %*% 符號
A <- matrix(c(4, 0, 5, -3, 1, 4, 2, -1, 0), nrow = 3, ncol = 3)
B <- matrix(c(2, 3, -1, 2, 1, 1, -5, 0, 4), nrow = 3, ncol = 3)
A %*% B

# 矩陣 M 與其反矩陣 M_inv 相乘可以得到一個單位矩陣
M <- matrix(c(4, 2, -7, -3), nrow = 2, ncol = 2)
M_inv <- solve(M)
M %*% M_inv
```

- 矩陣 M 與其反矩陣 M_inv 相乘可以得到一個單位矩陣I

5-5. 解線性聯立方程組

透過矩陣運算或 solve() 函數

$$AX = B$$

$$A^{-1}AX = A^{-1}B$$

$$X = A^{-1}B$$

- 寫出係數矩陣 A
- 寫出常數矩陣 B
- 找出係數矩陣A的反矩陣 A_inv
- 將 A_inv 與 B 相乘，即可得到X的解答$A^{-1}B$

```
# 解線性聯立方程組：透過矩陣運算
A <- matrix(c(2, 1, 3, 4, 3, -2, 5, 1, -4, 3, 1, -1, 1, -2, -1, 1), nrow = 4, ncol =
B <- matrix(c(15, -3, 20, 5), nrow = 4, ncol = 1)
A_inv <- solve(A)
x <- A_inv %*% B
x

# 解線性聯立方程組：透過 solve() 函數
A <- matrix(c(2, 1, 3, 4, 3, -2, 5, 1, -4, 3, 1, -1, 1, -2, -1, 1), nrow = 4, ncol =
B <- matrix(c(15, -3, 20, 5), nrow = 4, ncol = 1)
x <- solve(A, B)
x
```

特徵值與特徵向量

$AX = \lambaX$，\lamba是特徵值，X是特徵向量。 由$det(A - \lambaI) = 0$求解特徵值，再求特徵向量。

```
A <-matrix(c(0,1,0,-2,3,0,-3,3,1), nrow=3)
A

eigen(A)
```