Bioinformatics - R Introduction

Wei-Hao, Lee

2/21/2019

R 基礎教學

目錄

0. R installation
1. 變數 - Variable :
   - 變數指派
   - 移除變數
2. 運算子 - Operator :
   - Arithmetic Operator
   - Logical Operator
3. 變數類型與資料型別 :
   - 變數類型
     - Numeric
     - Integer
     - Logical
     - Character
   - 資料型別
     - One Dimension
       - Vector
       - Factor
     - Two Dimension (Next Class)
       - Matrix
       - Dataframe
     - Multi-Dimension (Next Class)
       - List
4. 流程控制(if)
5. 迴圈(for)
   - 基礎操作
   - 流程控制搭配`break`和`next`
6. Practice

---

0. R installation

R installation tutorial : 輕鬆學習R語言
而這個網頁也會是我們第一堂以及第二堂的參考資料，如果大家對於上課內容有疑惑的也可以到這邊參考。

助教的github : Go to github
這邊也會放入上課的講義，同時也會把上課操作的程式放在這裡提供大家參考，如果有問題也可以在issue中提問。

---

1. 變數 - Variable

變數主要用來存取任何合法資料，使用者可以方便地使用變數名稱來獲取或者使用資料。

```
# 將"偉豪"這個資料存入變數 teacher_assistant中
teacher_assistant <- "偉豪"

# 我們就可以利用 teacher_assistant 這個變數來獲取裡面的資料
print(teacher_assistant)
```

```
## [1] "偉豪"
```

變數指派

在上面的程式碼中，可以看到<-這個符號。這個符號用於將資料**賦予(assign)**給變數。以上述的例子，我們將"偉豪"這串文字賦值給變數teacher_assistant中。在其他程式語言中，比較常用=當作賦值的符號，而R語言也有支援這種=賦值符號。

```
# R語言中最常使用的方法，也是最道地的用法
TA_1 <- "乃文"

# 在其他程式語言中賦值的方法，在許多時候仍可以看到。
TA_2 = "嘉琪"

# 也是賦值的方法之一，但極不推薦使用。
"漢萱" -> TA_3

cat(TA_1, TA_2, TA_3)
```

```
## 乃文 嘉琪 漢萱
```

移除變數

當變數或者資料不再需要使用時，我們可以移除變數，讓這些資料不佔用電腦的記憶體。我們可以使用rm()這個函式將變數以及內部的資料給刪除。

```
TA_4 <- "玉潔"
print(TA_4)

## [1] "玉潔"

# 當變數 TA_4 不再需要使用時，可以將它刪除
rm(TA_4)

# 若再次呼叫變數 TA_4 則會得到下方的警告
TA_4

## Error in eval(expr, envir, enclos): object 'TA_4' not found
```

變數命名

變數在命名，盡量以明確為主要的命名方式，不要以代號的形式作為變數名稱。為了程式的可讀性，我們會盡量在命名時花點心思，讓我們日後再次編輯程式時，不會感到困惑。

```
# 最不推薦的方式，更不要以a,b,c,d為順序去做命名
a <- "MAPKKK"

# 明確的指出 "TP53" 是我們主要的目標基因
target_gene <- "TP53"
```

---

2. 運算子 - Operator

算術運算子 - Arithmetic Operator

算術運算子就是我們幼稚園教的加減乘除+、-、*、/。而R語言中有提供更多的算術運算子像是次方^ or **以及獲取餘數%%。*(補充說明：在不同程式語言中，他們有各自定義的運算子，即使符號相同其代表的意義也不同，所以在使用不同程式語言時不妨看看其運算子的定義)*

```
x <- 4
y <- 3

# 4 + 3
x + y
```

```
## [1] 7
```

```
# 4 - 3
x - y
```

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

```
# 4 * 3
x * y
```

```
## [1] 12
```

```
# 4 / 3
x / y
```

```
## [1] 1.333333
```

```
# 4 / 3 = 3...1 取餘數 1
x %% y
```

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

```
# 4 * 4 + 3 * 3 = 25
x^2 + y**2
```

```
## [1] 25
```

*補充說明：以次方為例，python中^是bitwise operator，而非代表次方。另外在其他程式語言像Java中並沒有支援** or ^等運算子*

邏輯運算子 - Logical Operator

邏輯運算子包含大於>、小於<、大於等於>=、小於等於<=、等於==、不等於!=、交集&&、聯集||。透過兩者數值的比較之後告訴你布林值(boolean：TRUE or FALSE)。

```
weight_a <- 100
weight_b <- 20

# 100 > 20 : TRUE
weight_a > weight_b
```

```
## [1] TRUE
```

```
# 100 < 20: FALSE
weight_a < weight_b
```

```
## [1] FALSE
```

```
# 100 >= 20: TRUE
weight_a >= weight_b
```

```
## [1] TRUE
```

```
# 100 <= 20: FALSE
weight_a <= weight_b
```

```
## [1] FALSE
```

```
# 100 == 20: FALSE
weight_a == weight_b
```

```
## [1] FALSE
```

```
# 100 != 20: TRUE
weight_a != weight_b
```

```
## [1] TRUE
```

```
# 兩個敘述都為 TRUE 時，才會回傳 TRUE
# 若其中一者為 FALSE 時，則回傳 FALSE

# 100 > 30 and 20 < 200 : TRUE
weight_a > 30 && weight_b < 200
```

```
## [1] TRUE
```

```
# 100 < 30 and 20 < 200 : FALSE
weight_a < 30 && weight_b < 200
```

```
## [1] FALSE
```

```
# 其中一個敘述為 TRUE 則不管另一個是否為真都會回傳 TRUE

# 100 < 30 or 20 < 200 : TRUE
weight_a < 30 || weight_b < 200
```

```
## [1] TRUE
```

---

3. 變數類型與資料型別 - Data Type

在上述的舉例中我們並沒有明確告訴你們資料的**類型或型別(type)**，因為我們可以一眼看出他是文字還是數值，但電腦並無法直接判斷，因此我們需要告訴他是什麼類型。然而現今許多程式語言都具有自動判斷的功能，因此有時候我們就會忽略掉變數的類型，但是變數的型別在某些時候相當的重要，因此我們需要認識一些基本的資料型別。

變數類型

我們可以利用函式`class()`查詢變數的類型。

```
class("偉豪")

## [1] "character"

class(30)

## [1] "numeric"

# 留意此處的 30 有加雙引號
class("30")

## [1] "character"
```

Numeric

在R語言中，任何整數或者小數(浮點數)都會直接歸類為`numeric`

```
# 當我們輸入整數時
right_angle <- 90
class(right_angle)

## [1] "numeric"

# 當我們輸入小數時
angle <- 24.5
class(angle)

## [1] "numeric"
```

補充說明：在其他程式語言中，整數(integer)和浮點數(float)是兩種不同的資料型別，為什麼要區別主要原因是不同的資料型別會戰有不同的記憶體空間。

Integer

在R語言中並不會常使用整數這個型別，主要原因是numeric已經有包含整數的運算，所以無須特別將`numeric`轉成`integer`。但在其他程式語言像:C, C++, Java都會時常使用整數型別，因此特別介紹。

```
# 要宣告成整數型態要在整數後面加上"L"
age <- 24L
class(age)

## [1] "integer"
```

```r
# 另一種方法
age <- as.integer(24)
class(age)

## [1] "integer"
```

Logical

在所有語言中都會有布林值(boolean)，而布林值其實就是TRUE和FALSE。在其他語言中，比較常看到boolean或bool代表其型態。在R語言中則是使用logical代表了是與否。布林值在流程控制中相當的重要，而且時常搭配邏輯運算子去做真假值的判斷。

```r
isGameOver <- TRUE
class(isGameOver)

## [1] "logical"

isTimeOut <- FALSE
class(isTimeOut)

## [1] "logical"

# 也可以只使用 T or F 代表 TRUE and FALSE
isStupid <- F
isHarmful <- T

cat(isStupid, isHarmful)

## FALSE TRUE
```

補充說明:在許多程式語言中，整數0代表FALSE，大於零的整數皆代表TRUE

Character

Character其實就是我們熟知的文字，當我們要告訴電腦我是要輸入數字還是文字時，我們需要加上單引號(’)或者雙引號(“)就能代表我要輸入的是文字。

```r
gene_length <- "2500"
class(gene_length)

## [1] "character"

gene_length <- 2500
class(gene_length)

## [1] "numeric"
```

```r
teacher_assistant <- 'Wei-Hao'
class(teacher_assistant)
```

```
## [1] "character"
```

補充說明：在其他程式語言中，其實文字可以細分為字串(string)以及字元(character)，兩者代表著不同的資料型態，也因為有兩種不同的型態，通常字元會使用單引號('a')而字串會使用雙引號("Wei-Hao")。

資料型別 – One Dimension

這別雖說是資料型別，其實也可以視為變數型別，但他只是含有許多資料的型別，像是我們知道的向量，他就是含有許多數值的一個資料型別，但細看他還是由最基本的數值所組成的大型別。

Vector

vector就是向量，他可以包含許多的數值或者文字，是一個相當常用的類型。在宣告向量時我們會使用`c()`將內容物給框住，請看下方範例。

```r
edges <- c(3,4,5)
print(edges)
```

```
## [1] 3 4 5
```

我們可以利用索引(index)去搜索你像要看的第幾個元素，其搜索指令為`[index]`，index的範圍由1到向量的長度。

```r
# series is a vector which contains 1 to 9
series <- c(1,2,3,4,5,6,7,8,9)
print(series)
```

```
## [1] 1 2 3 4 5 6 7 8 9
```

```r
# using [index] to search nth elements in the vector
print(series[5])
```

```
## [1] 5
```

```r
# using length() function to get length of vector
length_of_series <- length(series)
print(length_of_series)
```

```
## [1] 9
```

```r
# moreover, you can search part of vector using ":"
series[3:length_of_series]
```

```
## [1] 3 4 5 6 7 8 9
```

值得一提，當我們故意在向量中同時寫入數值以及文字或布林值，會發生什麼事情呢？

```
# 會被統一的轉成文字類型
information <- c("Wei-Hao", 24, TRUE)
print(information)

## [1] "Wei-Hao" "24"      "TRUE"

# information 中的第二個元素24被轉成了character
class(information[2])

## [1] "character"
```

更多的向量操作會在下一堂課中進行操作。

Factor

因素向量，他是一種相當特別的型別，有別於向量，他另外帶有曾級(Levels)的額外資訊在內。

```
severity_vector <- c("normal", "stage i", "stage ii", "stage iii", "stage
iv")
print(severity_vector)

## [1] "normal"    "stage i"   "stage ii"  "stage iii" "stage iv"

# 留意輸出結果的資訊多了一欄Levels
severity_factor <- factor(severity_vector)
print(severity_factor)

## [1] normal    stage i   stage ii  stage iii stage iv
## Levels: normal stage i stage ii stage iii stage iv

# 此時我們可以而外加入一些參數，告訴電腦嚴重的程度，留意輸出結果
severity_factor <- factor(severity_vector, ordered = TRUE,
                          levels = c("normal", "stage i", "stage ii", "stage
iii", "stage iv"))
print(severity_factor)

## [1] normal    stage i   stage ii  stage iii stage iv
## Levels: normal < stage i < stage ii < stage iii < stage iv

#
假設我們沒有給予Levels的資訊，但告訴電腦我們要有程度的排名，他會按照字母順序幫你做排序
cancer_type <- c("Lung", "Breast", "Brain", "Colorectal")
cancer_factor <- factor(cancer_type, ordered = TRUE)
print(cancer_factor)
```

```
## [1] Lung       Breast     Brain      Colorectal
## Levels: Brain < Breast < Colorectal < Lung
```

factor是一個相對難理解以及操作的資料型態，因此通常會使用向量，如果有需要時再轉為因素向量，在我們後續課程中會使用到因素向量做繪圖以及基因表象量差異計算。

---

4. 流程控制 (Control flow)

流程控制主要用途就是要來判斷是否該執行以下的程式碼，因此需要給予至少一個條件式或布林值，而在程式語言中最常使用的就是if或者if ... else 或者當條件多個時可以使用 if ... else if ... else。其實if...else。

```
age <- 13

# 年齡是否大於18歲
isOver_18 <- age >= 18
if (isOver_18) {
    # 如果大於等於18歲就會執行 {} 內的程式
    print("歡迎觀賞")
}

# 由於isOver_18判斷為FALSE所以不會執行內部程式
print(isOver_18)
```

```
## [1] FALSE
```

```
# five years later
age <- age + 5

# 年齡是否大於18歲
if (age >= 18) {
    # 如果有大於等於18就會執行 {} 內的程式
    print("歡迎觀看")
} else {
    # 如果未滿十八則會執行 else 後面 {} 內的程式
    print("禁止觀看")
}
```

```
## [1] "歡迎觀看"
```

```
# sixty years later
age <- age + 60

# 年齡是否大於18歲且小於六十歲
if (age >= 18 && age < 60) {
    # 如果有大於等於18且小於60就會執行 {} 內的程式
    print("歡迎觀看")
```

```
} else if (age >= 60){ # 檢查年紀是否超過60
    # 如果大於60歲則會執行 else if 後面 {} 內的程式
    print("太過刺激請勿觀看")
} else {
    # 當 age >= 18 和 age >= 60 都為FALSE的時候，就會最後else的程式內容
    print("禁止觀看")
}
```

```
## [1] "太過刺激請勿觀看"
```

```
# 改寫上方的程式碼，利用巢狀結構來改寫
if (age >= 18) {
    if (age >= 60) {
        print("太過刺激請勿觀看")
    } else {
        print("歡迎觀看")
    }
} else {
    print("禁止觀看")
}
```

```
## [1] "太過刺激請勿觀看"
```

---

5. 迴圈 (loop)

基本操作

迴圈就是讓電腦不斷地做重複的事情，直到條件達成。

```
# 產生1到10的序列
one_to_ten <- 1:10

# 利用for迴圈印出1到10
for (i in one_to_ten) {
    print(i)
}
```

```
## [1] 1
## [1] 2
## [1] 3
## [1] 4
## [1] 5
## [1] 6
## [1] 7
## [1] 8
## [1] 9
## [1] 10
```

值得留意的是R語言的for迴圈通常需要給定一個可以迭代的物件像是向量，讓他可以不斷的拜訪內部的成員。

```
# 如果想要找找看向量中是否有某個特定數值或文字該怎麼辦

# 產生1到5的序列
one_to_twenty <- 1:5

# 假設我們要找找看裡面是否有3
for(number in one_to_twenty) {
    if (number == 3) {
        cat(number, ":",  "我找到3了\n")
    } else {
        cat(number, ":", "還沒找到\n")
    }
}
```

```
## 1 : 還沒找到
## 2 : 還沒找到
## 3 : 我找到3了
## 4 : 還沒找到
## 5 : 還沒找到
```

流程控制搭配break和next

在迴圈中，有時候我們會碰到一些條件，讓我們想要跳脫迴圈或者忽略掉它，因此有了break和next這兩個關鍵字

```
# break 可以使迴圈跳出，但只會跳出一層
for (i in 1:10) {
    # 當i==5的時候會跳出此迴圈
    if (i == 5) {
        break
    }
    print(i)
}
```

```
## [1] 1
## [1] 2
## [1] 3
## [1] 4
```

```
# nested for loop with break
for (i in 1:3) {
    for (j in 4:6) {
        if (j == 5) {
            break
```

```
        }
        cat(i, "-", j, "\n")
    }
}
```

```
## 1 - 4
## 2 - 4
## 3 - 4
```

```
# next 可以使迴圈跳過後續要做的程式，並開始下一輪迴圈
for (i in 1:10) {
    # 當i==5的時候會略過5
    if (i == 5) {
        next
    }
    print(i)
}
```

```
## [1] 1
## [1] 2
## [1] 3
## [1] 4
## [1] 6
## [1] 7
## [1] 8
## [1] 9
## [1] 10
```

```
# nested for loop with next
for (i in 1:3) {
    for (j in 4:6) {
        if (j == 5) {
            next
        }
        cat(i, "-", j, "\n")
    }
}
```

```
## 1 - 4
## 1 - 6
## 2 - 4
## 2 - 6
## 3 - 4
## 3 - 6
```

補充說明：在其他語言中，continue等同於next的功能

Practice

簡單. 利用 for loop 做出九九乘法表

```
## 1 * 1 = 1    1 * 2 = 2   1 * 3 = 3   1 * 4 = 4   1 * 5 = 5   1 * 6 = 6   1
* 7 = 7   1 * 8 = 8   1 * 9 = 9
## 2 * 1 = 2    2 * 2 = 4   2 * 3 = 6   2 * 4 = 8   2 * 5 = 10  2 * 6 = 12  2
* 7 = 14  2 * 8 = 16  2 * 9 = 18
## 3 * 1 = 3    3 * 2 = 6   3 * 3 = 9   3 * 4 = 12  3 * 5 = 15  3 * 6 = 18  3
* 7 = 21  3 * 8 = 24  3 * 9 = 27
## 4 * 1 = 4    4 * 2 = 8   4 * 3 = 12  4 * 4 = 16  4 * 5 = 20  4 * 6 = 24  4
* 7 = 28  4 * 8 = 32  4 * 9 = 36
## 5 * 1 = 5    5 * 2 = 10  5 * 3 = 15  5 * 4 = 20  5 * 5 = 25  5 * 6 = 30  5
* 7 = 35  5 * 8 = 40  5 * 9 = 45
## 6 * 1 = 6    6 * 2 = 12  6 * 3 = 18  6 * 4 = 24  6 * 5 = 30  6 * 6 = 36  6
* 7 = 42  6 * 8 = 48  6 * 9 = 54
## 7 * 1 = 7    7 * 2 = 14  7 * 3 = 21  7 * 4 = 28  7 * 5 = 35  7 * 6 = 42  7
* 7 = 49  7 * 8 = 56  7 * 9 = 63
## 8 * 1 = 8    8 * 2 = 16  8 * 3 = 24  8 * 4 = 32  8 * 5 = 40  8 * 6 = 48  8
* 7 = 56  8 * 8 = 64  8 * 9 = 72
## 9 * 1 = 9    9 * 2 = 18  9 * 3 = 27  9 * 4 = 36  9 * 5 = 45  9 * 6 = 54  9
* 7 = 63  9 * 8 = 72  9 * 9 = 81
```

普通. 利用 if and for 做出三角形 (範例：高度=10)

```
##           *
##          ***
##         *****
##        *******
##       *********
##      ***********
##     *************
##    ***************
##   *****************
##  *******************
```

普通. 目前生資國只有131, 23, 11, 7,
1等幣值，請寫出一個程式，計算出怎們樣換錢能到最少的紙鈔跟硬幣。ex.
我有2019元，請問怎麼換？

```
## $131 :   15
## $23  :   2
## $11  :   0
## $7   :   1
## $1   :   1
```

補充資料：print(), cat(), paste() 使用方式

print()這個函式只能放入一個變數，所以在使用時通常會搭配paste()去做使用。而cat()這個函式適合多個變數帶入，如果有認真觀察，上面的例子，就會發現我使用上的區別囉！

```
teacher_assistant <- "Wei-Hao Lee"

# 在只有一個變數的情況下 cat() 和 paste() 無差別
print(teacher_assistant)

## [1] "Wei-Hao Lee"

cat(teacher_assistant)

## Wei-Hao Lee

# 如果給予 print() 超過一個變數時
print(teacher_assistant, "阿不就好棒棒")

## Warning in print.default(teacher_assistant, "阿不就好棒棒"): NAs introduced
## by coercion

## Error in print.default(teacher_assistant, "阿不就好棒棒"): invalid 'digits'
argument
```

因此在多個變數時我們可以使用 cat() 或者
paste()，然而這兩個函式使用上有些差異，cat() 跟 print()
很像可以將變數裡的內容印出，但是 paste()
則是將變數裡的內容黏在一起形成文字(字串)，通常我們會使用一個變數將合併完的文字儲存，而 cat() 則無法用變數把內容保存。

```
# cat() 會直接印出來，但是無法用變數將資料保存下來
cat(teacher_assistant, "阿不就好棒棒")

## Wei-Hao Lee 阿不就好棒棒

# 不會告訴你錯誤，並且會印出結果
statement <- cat(teacher_assistant, "阿不就好棒棒")

## Wei-Hao Lee 阿不就好棒棒

# 但是再次使用變數則會發現是空的
print(statement)

## NULL

# paste() 會將變數內的東西合併在一起，並且可以用變數保存
paste(teacher_assistant, "阿不就好棒棒")
```

```
## [1] "Wei-Hao Lee 阿不就好棒棒"
```

```
statement <- paste(teacher_assistant, "阿不就好棒棒")
print(statement)
```

```
## [1] "Wei-Hao Lee 阿不就好棒棒"
```

眼尖的你們是否發現使用paste()和cat()中間都會有一個空白鍵呢？原因是在函式內部設定已經幫你在文字中加入一個空白鍵，如果我們想要更改的話可以在函式的括弧中加入sep="你想要的任何東西"。

```
cat("HEHE!!", "HAHA", sep="***你想要的任何東西***")
```

```
## HEHE!!***你想要的任何東西***HAHA
```

```
paste("ID", "000123", sep=":")
```

```
## [1] "ID:000123"
```

```
# 常見的特殊字元"\n", "\t"

# "\n" newline 換新的一行，是否發現"切斷了"前面有空白呢？為什麼？
cat("我被", "\n", "切斷了")
```

```
## 我被
##  切斷了
```

```
# "\t" tab鍵的功能
cat("我被", "\t", "拉長了")
```

```
## 我被     拉長了
```

E-mail : steve24563@gmail.com