



R Programming Foundation

- Variable
- Data type
- Data structure
- Function
- control flow

Common Errors

- เขียนโปรแกรมยังไม่จบ เทbinเป็นเครื่องหมาย + ใน console แก้ด้วยการกด ESC
- Object 'xxx' not found คือยังไม่ประกาศตัวแปร หรือพิมพ์ชื่อผิด ลอง check ที่ environment หรือลองประกาศใหม่
- Could not find function "xxx" คือหาชื่อฟังก์ชันไม่เจอ แก้ด้วยการดูว่าเขียนชื่อฟังก์ชันถูก หรือป่าว หรือเรียกใช้ `library()`

Reading files to R

- ติดตั้ง library

```

install.packages(c("readr",
                  "readxl",
                  "googlesheets4",
                  "jsonlite",
                  "dplyr",
                  "sqldf",
                  "RSQLite"))

library(readr)
library(readxl)
library(googlesheets4)
library(jsonlite)
library(dplyr)
library(sqldf)
library(RSQLite)

```

- อ่านไฟล์ .txt คือจะค้นคอลัมน์ด้วย write space ให้เรียกใช้ `library(readr)` ก่อน และใช้ `read_table("name files.txt")`
- อ่านไฟล์ .csv คือค้นด้วยคอมม่า `read_csv("name files.csv")`
- อ่านไฟล์ excel ก่อนอื่นเรียก `library(readxl)` และใช้ `read_excel("names.xlsx", sheet=๑๖๘.หรือชื่อคอลัมน์")` หรือจะเรียกที่ละหลาย ๆ sheet ตามโค้ดด้านล่าง

```

result <- list()

for (i in 1:3) {
  result[[i]] <- read_excel("name.xlsx", sheet=i)
}

```

- Google Sheet

```

library(googlesheets4)

url <- "link from Google sheet au /edit... ออก"

gs4_deauth() # คือการบอกว่าลิ้งค์ที่ได้มาเป็น public not log in

```

```
read_sheet(url, sheet="name sheet")
```

- JSON

```
library(jsonlite)
```

```
data <- data.frame(fromJSON("names.json")) # เปลี่ยน list เป็น data
```

- bind_row คือการนำหลาย ๆ ตารางมาต่อกันจากด้านล่าง แต่เมื่อตอนไขว่า ชื่อคอลัมน์และประเภทของคอลัมน์ต้องเหมือนกัน

```
library(dplyr)  
bind_row(ชื่อตาราง1, ชื่อตาราง2, ...)
```

```
# หรือถ้าคอลัมน์บันเยอะก็ เก็บในลิส และนำชื่อลิสไปใส่ใน bind_row(name list)
```

- bind_cols() เป็นการต่อคอลัมน์ ไม่เหมือนกับการเขียน join ไม่ต้องใช้ PK=FK แค่เอาตารางมาต่อ กันเฉย ๆ

```
library(dplyr)  
bind_cols(ชื่อตาราง 1, ชื่อตาราง 2, ...)
```

Variable

คือ วิธีการประกาศตัวแปร ในภาษา R จะใช้เครื่องหมายเท่ากับเป็น `<-` หรือ `->`

- ชื่อห้ามขึ้นต้นด้วยตัวเลข
- วิธีลบตัวแปร ให้ใช้คำสั่ง `rm(ชื่อตัวแปร)`

** Ctrl L เคลียร์หน้าจอ Console

Data type

การเช็คประเภทของข้อมูล `class(ชื่อตัวแปร)`

- **numeric** คือตัวเป็นตัวเลข ถ้า 1L ประเภทก็ยังคงเป็น numeric
- **character (text, string)**

การเชื่อมสองข้อความเข้าด้วยกันจะใช้คำสั่ง paste

```
paste0("Hello", "world") #'Helloworld'
paste("Hello", "world") #'Hello world'
```

#ไม่สามารถเชื่อมกันด้วย + เนื่องจากภาษาอื่น

- **boolean** คือ TRUE, FALSE ซึ่งส่วนมากจะใช้ทำงานกับการใช้เกณฑ์สมการ เช่น $1+1 == 2$ คำตอบจะเป็น TRUE และถ้าจะเป็นจาก TRUE เป็น FALSE ให้ใช้เครื่องหมาย ! เช่น !(TRUE) คำตอบจะได้ FALSE
- **date** "YYYY-MM-DD" ถ้าเราพิมพ์ไปตรงๆแบบนี้ ประเภทของข้อมูลจะยังคงเป็น character เราจะต้องใช้คำสั่ง `as.Date(วันที่)` เพื่อเปลี่ยน text เป็น วันที่

```
today_date <- as.Date("2023-12-04")
print(today_date)
print(class(today_date))

# [1] "2023-12-04"
# [1] "Date"
```



เพรำณบັນອັນນີ້ສຳຄັນ ກາຣຈະປຶກປະເກທຂ້ອມູລໃຫ້ໃຊ້ຄຳສັ່ງ as.character ປຶກທີ່ຕ້ອງການ
ປຶກປະເກທຂ້ອມູລໃຫ້ໃຊ້ຄຳສັ່ງ `as.character(ช່ອຕັວແປຣ)`

- **factor** ຄ້າຍໆ ກັບ array ໃນ google sheet ຮັ້ອຈະເຮັດວ່າເປັນລັສກິໄດ້ ຄັ້ງຈະເປັນກາຣເກີບຂ້ອມູລຫາຍໆຕັວ ຮັ້ອໃຊ້ໃນກາຣແບ່ງກຸ່ມ (ສ້າງ vector) ແຕ່ຂ້ອມູລທີ່ເກີບຕ້ອງເປັນປະເກດເດືອວັກນ

```
gender <- c("male", "female", "kid")

gender <- factor(gender)
```

```
gender  
  
# male female kid  
# Levels:
```

- การเปลี่ยน Data Type

```
char_x <- as.character(x) # character  
num_x <- as.numeric(x) # numeric  
  
as.logical(0) # FALSE  
as.logical(1) # TRUE  
as.numeric(TRUE) # 1
```

Data structure

- **vector (single data type)**

สามารถสร้างได้ดังรูป โดย c คือ concat คือการเอาข้อมูลมาต่อๆ กันนั้นเอง **เก็บข้อมูลได้แค่ 1 ประเภท (Data Type)**

```
scores <- c(88, 90, 50, 65, 76)  
scores  
  
# 88 90 50 65 76
```

สามารถเช็คค่า > หรือ < ได้ภายในบรรทัดเดียว เรียกว่า **vectorization** คือไม่ต้องเขียน loop หรือ จับค่านวนเลข + - * / ก็ได้

```
scores > 80  
  
# TRUE TRUE FALSE FALSE FALSE
```

การหา **sum, mean, min, max, length**

```
print(sum(scores))
print(mean(scores))
print(min(scores))
print(max(scores))
print(length(scores))

# [1] 369
# [1] 73.8
# [1] 50
# [1] 90
# [1] 5
```

การ **subset vector** หรือการใส่ชื่อให้กับ **vector Index** จะเริ่มด้วยเลข 1 มันจะไม่เหมือนกับ ภาษาอื่น ส่วนการเรียกใช้จะเรียกด้วยชื่อหรือ Index ก็ได้ เช่น scores[1] จะได้ join:88 หรือจะเรียกตามเงื่อนไข ดังรูป 2

```
scores <- c(
  join = 88,
  minnie = 90,
  david = 50,
  marry = 65,
  anna = 76)

scores

# join: 88 minnie: 90 david: 50 marry: 65 anna: 76
```

```
scores[scores > 80]

# join: 88 minnie: 90
```

การ **update value** การเปลี่ยนค่าในลิส เช่นจะเปลี่ยนคะแนนของ david เป็น 60

```

scores[3] <- 60
scores

# join: 88minnie: 90david: 60mary: 65anna: 76

```

และถ้าอยากรู้ตำแหน่งของคนที่ได้คะแนนมากที่สุด ให้ใช้คำสั่ง `which.max(ชื่อตัวแปร)` หรือ คะแนนน้อยสุดก็ได้

```

which.max(scores)

# minnie: 2

```

หรือจะแบบนี้ก็ได้

```

friends <- c("toy", "john", "mary", "top", "janny")
paste("hi", friends)

# 'hi toy' 'hi john' 'hi mary' 'hi top' 'hi janny'

```

- matrix (single data type) คือ vector ที่มีสอง dimensions

ตัวแรกของ matrix คือจำนวนตัวเลขจาก...ถึง... ซึ่งสองคือการเซตจำนวนแกร่งที่ต้องการ หรือจะเปลี่ยนเป็นจำนวนคอลัมน์ก็ได้ สิ่งเกตว่าเลขจะว่างตามคอลัมน์ลงมา ถ้าอยากรู้เลขว่างตามแกร่ง ให้ใส่ `byrow=TRUE` ในช่องที่สาม และสามารถทำ vectorization ได้เลย

```

m1 <- matrix(1:10, nrow=2, byrow= TRUE)
m1

# A matrix: 2 × 5 of type int
1   2   3   4   5
6   7   8   9   10

```



คำสั่ง `seq` คือการสร้างตัวเลขจาก.. ถึง .. โดยกำหนดค่ากี่เพิ่มขึ้นໄฉ่ `seq(1, 100, 5) #`
`1. 6. 11. 16. 21. 26. 3. 36 . 41. 46. 51. 56. 61. 66. 71. 76. 81. 86. 91. 96`



การคุณ matrix ก็สามารถทำได้แต่ต้องเขียนแบบนี้ `m1 %*% m2` การหา dimention ของ matrix ก็หาได้จาก `dim(m1)`

- **list**

การสร้าง list ใน R ฝั่งขวากอง key จะเป็นข้อมูลประเภทไหนก็ได้ (**Key = value**) ดังรูป

```
customer01 <- list(  
  name = "toy",  
  age = 35,  
  city = "Bangkok",  
  favorite_films = c("Dark Knight", "The Marvels")  
)
```

การดึงข้อมูลในลิสจะใช้เครื่องหมาย [] ถ้าใช้ [[]] จะดึงเฉพาะข้อมูลฝั่งขวา

```
customer[["name"]] # 'toy'  
customer["name"] # $name = 'toy'  
customer[["favorite_films"]][2] # 'The Marvels'
```

การสร้าง nested lists คือการมีลิสมากกว่าหนึ่งลิสเก็บไว้ในตัวแปรเดียว ดังรูป

```
customer01 <- list(  
  name = "toy",  
  age = 35,  
  city = "Bangkok",  
  favorite_films = c("Dark Knight", "The Marvels")  
)  
  
customer02 <- list(  
  name = "top",  
  age = 22,  
  city = "Chiangrai",  
  favorite_films = c("Far From Away", "Caption Ba")  
)
```

```
# nested lists  
list_customer <- list(customer01, customer02)
```

ส่วนวิธีการดึงข้อมูลก็คล้ายๆกันกับแบบแรกคือการใช้ [] แต่ต้องดูตำแหน่ง index ให้ดี

```
list_customer[[1]][["name"]] # 'toy'
```

- **dataframe**

คือ ตารางใน excel นั้นแหล่ะ มีแก้ว มีคอลัมน์ ส่วนการสร้างเราจะใช้คอมเซ็ปกี่ว่า (**column = values**)

```
df <- data.frame(  
  id = 1:5,  
  name = c("toy", "john", "mary", "jane", "anna"),  
  age = c(28, 30, 31, 22, 25),  
  movie_lover = c(T, T, F, T, T)  
)  
df
```



```
# A data.frame: 5 × 4  
id   name    age movie_lover  
<int> <chr>   <dbl>   <lgl>  
1   toy     28     TRUE  
2   john    30     TRUE  
3   mary    31     FALSE  
4   jane    22     TRUE  
5   anna    25     TRUE
```

การ **subset** ข้อมูลจาก dataframe `df[ดึงแก้ว, ดึงคอลัมน์]`

```
df[ c(1, 3, 5), "age"] # 28 31 25  
  
df$name  
df[["name"]] # 'toy' 'john' 'mary' 'jane' 'anna'
```

```

# 'toy' 'john' 'mary' 'jane' 'anna'

df[ df$movie_lover, ]
# A data.frame: 4 × 4
  id   name     age movie_lover
  <int> <chr>    <dbl>   <lgl>
1     1   toy     28   TRUE
2     2   john    30   TRUE
4     4   jane    22   TRUE
5     5   anna    25   TRUE

```

หรือสามารถดึงตามเงื่อนไข และมากกว่าสองเงื่อนไขได้ เช่นด้วยและ (&) หรือ ()

```

df[ (df$age >= 30) & (df$movie_lover), ]
# A data.frame: 1 × 4
  id   name     age movie_lover
  <int> <chr>    <dbl>   <lgl>
2     2   john    30   TRUE

```

หรือสามารถ subset โดย `subset` ส่องคอลัมน์ติดกันเวลาเรียกใช้งาน จะแสดงผลพร้อมกัน ด้วย function > `names`

```

names(ages) <- friends
ages
# Wan Top Ink
# 28 30 27

ages["Top"]
# Top
# 30

```

การสร้างคอลัมน์ใหม่ ดังรูปคือสร้างคอลัมน์ city

```

df$city <- c(rep("BKK", 3), rep("LONDON", 2))
#rep = replicate
df

```

```
# A data.frame: 5 × 5
  id   name    age movie_lover city
  <int> <chr> <dbl>    <lgl>   <chr>
1 toy    28     TRUE     BKK
2 john   30     TRUE     BKK
3 mary   31     FALSE    BKK
4 jane   22     TRUE     LONDON
5 anna   25     TRUE     LONDON
```

ส่วนการลบคอลัมบ์ให้ใช้ค่า NULL ในการลบ

```
df$random <- 100

# remove column
df$random <- NULL
df

# A data.frame: 5 × 5
  id   name    age movie_lover city
  <int> <chr> <dbl>    <lgl>   <chr>
1 toy    28     TRUE     BKK
2 john   30     TRUE     BKK
3 mary   31     FALSE    BKK
4 jane   22     TRUE     LONDON
5 anna   25     TRUE     LONDON
```

การ Export ข้อมูลออกเป็น csv ไฟล์

```
## write csv file
write.csv(df, "movie.csv", row.names=FALSE)
```

ส่วนการ Import ไฟล์เข้ามา ใช้คำสั่ง `read.csv`

```
movie <- read.csv("movie.csv")
movie
```

```
# A data.frame: 5 × 5
  id   name    age movie_lover city
  <int> <chr> <int> <lgl>   <chr>
1 toy    28   TRUE    BKK
2 john   30   TRUE    BKK
3 mary   31  FALSE   BKK
4 jane   22   TRUE   LONDON
5 anna   25   TRUE   LONDON
```

สุดท้ายของฟังก์ชันในคอสเน็คต์ `head`, `tail`, `str`

```
head(movie, 2)

# A data.frame: 2 × 5
  id   name    age movie_lover city
  <int> <chr> <int> <lgl>   <chr>
1 1    toy    28   TRUE    BKK
2 2    john   30   TRUE    BKK

tail(movie, 2)

# A data.frame: 2 × 5
  id   name    age movie_lover city
  <int> <chr> <int> <lgl>   <chr>
4 4    jane   22   TRUE   LONDON
5 5    anna   25   TRUE   LONDON

# structure
str(movie)

# 'data.frame': 5 obs. of  5 variables:
# $ id        : int  1 2 3 4 5
# $ name      : chr  "toy" "john" "mary" "jane" ...
# $ age       : int  28 30 31 22 25
```

```
$ movie_lover: logi  TRUE TRUE FALSE TRUE TRUE  
$ city          : chr  "BKK" "BKK" "BKK" "LONDON" ...
```

การดึงคอลัมน์ทำได้สามแบบ we use [[]] to extract column as vector

- `data$col_name`
- `data[["col_name"]]`
- `data[[5]]`

Function

คือการเปลี่ยน Input ให้กลายเป็น output ให้อยู่รูปของ function ซึ่งทำให้เราสามารถเรียกใช้เมื่อไหร่ก็ได้ โดยสามารถเขียนในรูป simple ง่ายๆ

```
my_secret_formula <- function(start, end) {  
  (start + end) * end/2  
}  
  
my_secret_formula(1, 100)
```

การใส่ `return` ในฟังก์ชัน คือการที่เราจะคืนค่าในฟังก์ชันออกมาใส่ในตัวแปรที่เราต้องการได้และนำไปใช้ต่อ

```
add_two_num <- function(x,y) {  
  return(x+y)  
}  
  
result = add_two_num(2, 8)  
result
```

ตามโค้ดด้านบนคือรับค่าจากฟังก์ชันมาเก็บไว้ก่อนแล้วที่ตัวแปร `result` แต่ในภาษา R ไม่ต้องใส่ `return` ก็ได้ ได้คำตอบเห็นชอบกัน

ต่อมาเราสามารถเซต default argument ได้มากกว่าหนึ่ง argument ซึ่งการเขียนโค้ดคือเราจะใส่ค่าลงใน function ได้เลย

parameter vs. argument จากด้านล่าง `name, city` คือ **parameter** และ `john, london` คือ **argument**

```
greeting_city <- function(name="john", city="London") {  
  paste("Hi!", name, "Welcome to", city)  
}  
  
greeting_city()
```

The answer is 'Hi! john Welcome to London'

เราสามารถเปลี่ยนชื่อได้ผ่านวงเล็บในคำสั่ง `greeting_city` และตำแหน่งสามารถสลับกันได้

Take user input

การใส่ `input` จาก `user` ค่าที่ใส่มาແມ່ຈະเป็นตัวเลขหรือตัวอักษร คลาสที่ได้จะเป็น `character` เగ່ານັ້ນ
ถ้าจะทำเป็นชนิดอื่นต้องใส่ฟังก์ชัน `as.` เช่น

```
username <- readline("what is your name: ")  
password <- as.integer(readline("Your password is: "))
```

ฝึกเขียน **function** 3 ตัว

```
# add_two_nums บวกกันสองตัว  
  
add_two_nums <- function(val1, val2) {  
  val1 + val2  
}  
  
# cube() function ยกกำลัง  
  
cube <- function(base, power=3) {  
  return(base ** power)  
}  
  
# count_ball() function นับจำนวน  
  
balls <- c("red", "red", "blue", "green")
```

```
count_ball <- function(balls, color) {  
  sum(balls == color)  
}
```

Control Flow

- **if else** ใช้ในการควบคุมพฤติกรรมโปรแกรม

```
score <- 65  
  
if (score >= 80) {  
  print("A")  
} else if (score >= 70) {  
  print("B")  
} else if (score >= 60) {  
  print("C")  
} else if (score >= 50) {  
  print("D")  
} else {  
  print("F")  
}
```

the result is "C" ถ้าจะให้มีประสิทธิภาพให้เขียนใส่ในฟังก์ชัน แค่เปลี่ยน `print` เป็น `return`

หรือถ้าเราจดเขียนง่าย ๆ ดื้อ ใช้ `ifelse(condition, TRUE, FALSE)`

```
score <- 80  
  
ifelse(score >= 80, "passed", "failed")  
  
ifelse(score >= 90, "passed", ifelse(score >= 50, "ok", "enroll"))
```

- **for loop** (ไม่จำเป็นต้องใช้เก่าไหร่ใน R เพราะใน R มีคุณเช็ป **vectorization**)

```

fruits <- c("banana", "apple", "pineapple")

for (fruit in fruits) {
  print(toupper(fruit))
}

```

`apply()` function คือการกำช้ำทุกคอลัมน์ในตารางจากตัวอย่างคือมันจะแสดงผลลัพธ์ด้วยการหาค่าเฉลี่ยในทุกคอลัมน์

```

## แบบเดิมซึ่งเขียนໂຄດຍາವมาก
cal_mean_by_col <- function(df) {
  col_names <- names(df)

  for (i in 1:ncol(df)){
    avg_col <- mean(df[[i]])
    print(paste(col_names[i], ":", avg_col))
  }
}

## apply function
apply(mtcars, MARGIN = 2, mean) # margin = 1 คือการหาค่าเฉลี่ยตามแคลว

```

- **while loop be careful update count in while loop otherwise you will stuck in infinite loop**

```

count <- 0

while (1 > 0) {
  print("hi!")
  count = count + 1
  if (count == 5) {
    print("done!")
    break
}

```

```
    }  
}
```

คำตอบจะได้

```
[1] "hi!"  
[1] "hi!"  
[1] "hi!"  
[1] "hi!"  
[1] "hi!"  
[1] "done!"
```

SQL

ต้องดาวน์โหลด `library(sqldf)` และใช้คำสั่ง `sqldf` เพื่อเรียกดู

```
library(sqldf)  
library(readr)  
  
read_csv  
  
sql_query <- "select * from school where country = 'USA'"  
sqldf(sql_query) # เรียกใช้งาน
```

Read data from SQLite database

```
library(RSQLite)  
  
# connect to SQLite database (.db file)  
# 1. open connection  
conn <- dbConnect(SQLite(), "chinook.db")  
  
# 2. get data  
dbListTables(conn)  
dbListFIELDS(conn, "customers")
```

```
df <- dbGetQuery(conn, "select * from customers  
                           where country =  
  
# 3. close connection  
dbDisconnect(conn)
```

Save data in r

```
save.image(file = "data.RData") # คือการโหลดเดต้าที่อยู่ในไฟล์ที่เราจำลังโค้ด +  
load(file = "data.RData") # โหลดไฟล์ใน folder มาเก็บใน Environment  
  
saveRDS(business, file = "business.rds") # โหลดบางเดต้า single obj  
readRDS("business.rds") # อ่านไฟล์เข้ามา
```

Homework