

## R 102

Tags	Programming	R
Class		
Finished Yet?	<input checked="" type="checkbox"/>	
Knowledge	<a href="#">The Third Sprint: R</a>	

## What is Control Flow?

-Control Flow คือหนึ่งในหัวใจสำคัญในการเขียนโปรแกรม ใน R จะมี Control Flow สำคัญอยู่ 3 ตัว ได้แก่:

1. if
2. for
3. while

-หน้าที่ของ Control Flow คือการควบคุมพัฒนารูปแบบของโปรแกรมที่เราเขียน เช่น:

```
score <- 85
if (score >= 80) { print("Passed") }
else { print("Failed") }
```

[ถ้าคะแนนสอบมากกว่าหรือเท่ากับ 80 จะสอบผ่าน ถ้าคะแนนไม่ถึง 80 จะสอบตก]

- เราเขียน if เพื่อกำหนดเส้นทางในการทำงานของโปรแกรม (Path)

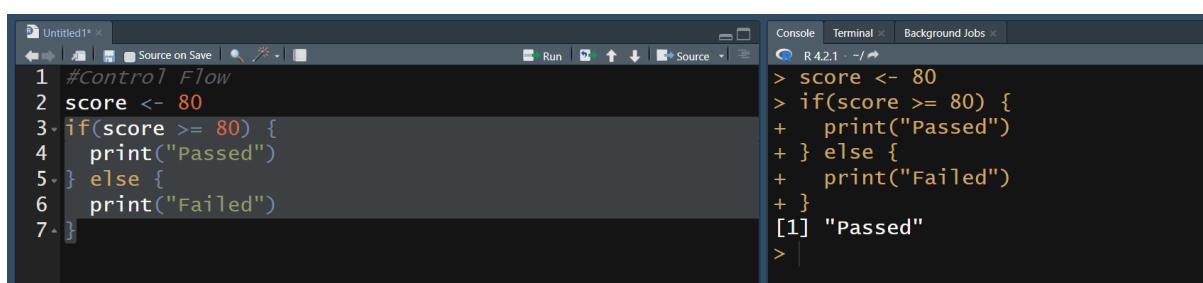
- เราสามารถใช้ for และ while ในการทำ loop ได้ ซึ่งการทำ loop ของ 2 ตัวนี้จะแตกต่างกัน

---

## Lesson 1: IF

- การเขียนเงื่อนไข IF เป็นพื้นฐานของภาษาคอมพิวเตอร์ทุกภาษา มีการเขียนเหมือนกันคือ:  
condition, if TRUE, else FALSE

- ตัวอย่างการเขียน IF:



The screenshot shows an RStudio interface with two panes. The left pane is titled "Untitled1" and contains the following R code:

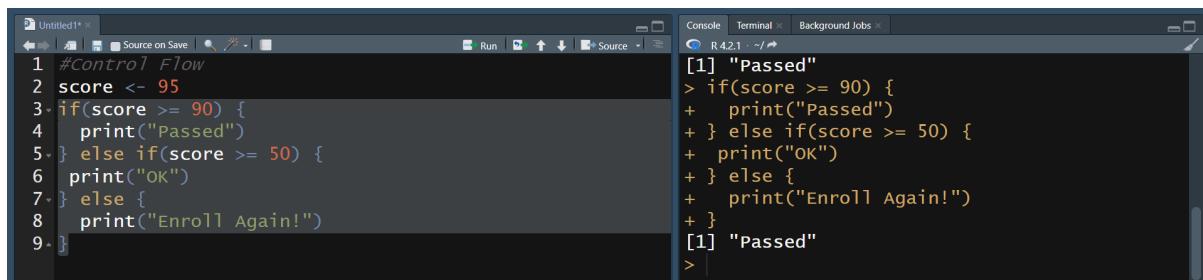
```
1 #Control Flow
2 score <- 80
3 if(score >= 80) {
4   print("Passed")
5 } else {
6   print("Failed")
7 }
```

The right pane is titled "Console" and shows the output of running the code:

```
> score <- 80
> if(score >= 80) {
+   print("Passed")
+ } else {
+   print("Failed")
+ }
[1] "Passed"
```

[ถ้า score มา กกว่าหรือเท่ากับ 80 แล้วจะสอบผ่าน ถ้าไม่ใช่ จะสอบตก \*อย่าลืมรัน score ← 80 ก่อน]

- เราสามารถทำ nested if (if ซ้อน if) ได้ เช่นกัน เช่น:



The screenshot shows an RStudio interface with two panes. The left pane is titled "Untitled1" and contains the following R code:

```
1 #Control Flow
2 score <- 95
3 if(score >= 90) {
4   print("Passed")
5 } else if(score >= 50) {
6   print("OK")
7 } else {
8   print("Enroll Again!")
9 }
```

The right pane is titled "Console" and shows the output of running the code:

```
[1] "Passed"
> if(score >= 90) {
+   print("Passed")
+ } else if(score >= 50) {
+   print("OK")
+ } else {
+   print("Enroll Again!")
+ }
[1] "Passed"
```

[ถ้า score มา กกว่าหรือเท่ากับ 90 และจะสอบผ่าน ถ้า score มา กกว่าหรือเท่ากับ 50 จะ OK ถ้าคะแนนต่ำกว่า 50 จะสอบตก]

---

## Lesson 2: ifelse()

- ifelse() เขียนเหมือน =IF() ใน Google Sheet: ifelse(condition, TRUE, FALSE)

-ยกตัวอย่างเช่น:

The screenshot shows an RStudio interface with two panes. The left pane contains R code:

```
3 if(score >= 90) {  
4   print("Passed")  
5 } else if(score >= 50) {  
6   print("OK")  
7 } else {  
8   print("Enroll Again!")  
9 }  
10  
11 ifelse(score >= 80, "Passed", "Failed")  
12
```

The right pane shows the R console output:

```
R 4.2.1 ~/  
> ifelse(score >= 80, "Passed", "Failed")  
[1] "Passed"  
>
```

[ถ้า score มากกว่าหรือเท่ากับ 80 แล้วจะสอบผ่าน ถ้าไม่ใช่ จะสอบตก]

-เราสามารถเขียน ifelse() ซ้อนกันได้ เช่น:

The screenshot shows an RStudio interface with two panes. The left pane contains R code:

```
5 } else if(score >= 50) {  
6   print("OK")  
7 } else {  
8   print("Enroll Again!")  
9 }  
10  
11 ifelse(score >= 80, "Passed", ifelse(  
12   score >= 50, "OK", "Enroll Again!")  
13 ))|  
14
```

The right pane shows the R console output:

```
R 4.2.1 ~/  
> ifelse(score >= 80, "Passed", ifelse(  
+   score >= 50, "OK", "Enroll Again!")  
+ ))  
[1] "Passed"  
>
```

[ถ้า score มากกว่าหรือเท่ากับ 90 แล้วจะสอบผ่าน ถ้า score มากกว่าหรือเท่ากับ 50 จะ OK ถ้า คะแนนต่ำกว่านั้นจะสอบตก]

## Lesson 3: FOR

-ในการใช้งานจริง เราจะแบ่งไม่ต้องเขียน For loop แบบ manual ใน R เพราะโค้ดใน R มีรับแบบ vectorization อุ่นๆแล้ว

-ตัวอย่างการรับ For loop:

The screenshot shows an RStudio interface with two panes. The left pane contains R code:

```
1 #For  
2 friends <- c("Toy", "John", "Mary", "Anna")  
3 for(friend in friends) {  
4   print(friend)  
5 }
```

The right pane shows the R console output:

```
R 4.2.1 ~/  
> friends <- c("Toy", "John", "Mary", "Anna")  
> for(friend in friends) {  
+   print(friend)  
+ }  
[1] "Toy"  
[1] "John"  
[1] "Mary"  
[1] "Anna"  
>
```

[สำหรับทุก ๆ ค่าใน friends ให้ print ออกมาทางหน้าจอ \*สังเกตว่าซื้อเพื่อบนจะໄล์กีฬะ 1 ชื่อจากบบลงล่าง]

[เราสามารถใช้ paste() ในการ Concatenate ค่าที่เป็นข้อความ (string) เข้าหากันได้]

-Vectorization ทำให้เราสามารถทำแบบนี้ได้โดยไม่ต้องเขียน For loop:

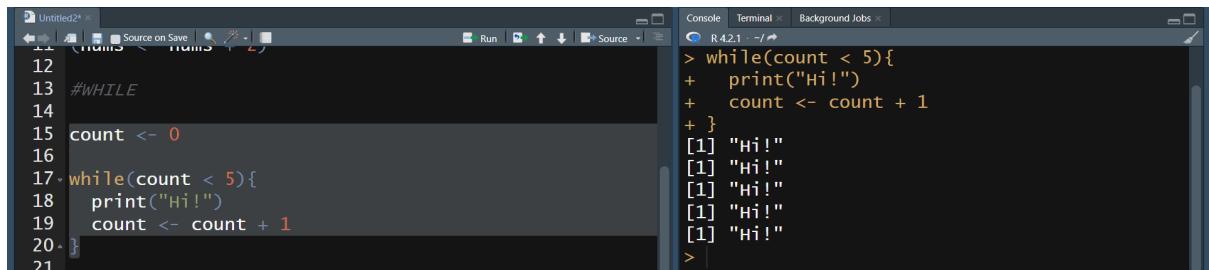
-เราสามารถทำ Vectorization กับตัวเลขได้เช่นกัน เช่น:

\*สำหรับภาษาโปรแกรมอื่น จะต้องเขียน For loop เพื่อบวกเลขทีละตัว

-ถ้าเราครองวงเล็บใน code เช่น ( nums <- nums +2 ) ตอนเรารันໂຄ้ด ผลลัพธ์จะแสดงแบบหน้าจอ กันที เช่น:

## Lesson 4: WHILE

-While loop จะต่างจาก For loop ตรงที่ loop จะวนไปเรื่อย ๆ จนกว่าจะเจอเงื่อนไขที่ทำให้ break loop เช่น:



The screenshot shows an RStudio interface with two panes. The left pane is a code editor with the following R code:12  
13 *#WHILE*  
14  
15 count <- 0  
16  
17 while(count < 5){  
18 print("Hi!")  
19 count <- count + 1  
20 }  
21The right pane is a terminal window titled "Console" with the output of the code:> while(count < 5){  
+ print("Hi!")  
+ count <- count + 1  
+ }  
[1] "Hi!"  
[1] "Hi!"  
[1] "Hi!"  
[1] "Hi!"  
[1] "Hi!"  
[1] "Hi!"  
>

[ถ้า count น้อยกว่า 5 โค้ดจะ print คำว่า Hi! จนกว่า count จะเท่ากับ 5]

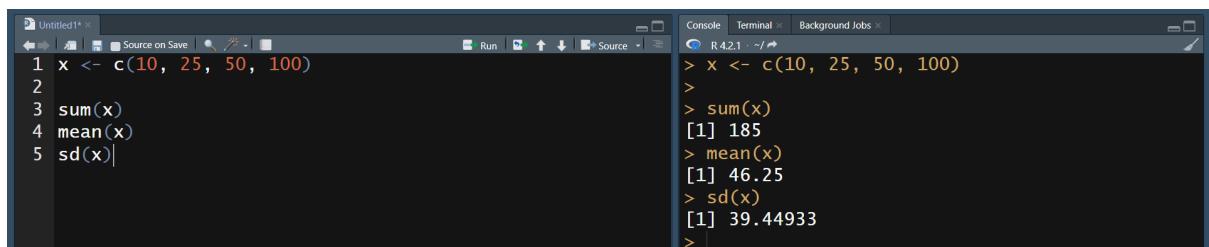
\*ถ้าติดอยู่ใน infinite loop เราสามารถหยุดการรันโค้ดได้ด้วยการกดกีร์ไอค่อน stop สีแดงตรงหน้าต่างคอนโซล

---

## Lesson 5: Function

-function เป็น core ในการเขียนโปรแกรม [input → f(x) → output]

-เราสามารถเขียน function เพื่อ manipulate dataset ของเราได้ เช่น:



The screenshot shows an RStudio interface with two panes. The left pane is a code editor with the following R code:1 x <- c(10, 25, 50, 100)  
2  
3 sum(x)  
4 mean(x)  
5 sd(x)|The right pane is a terminal window titled "Console" with the output of the code:> x <- c(10, 25, 50, 100)  
>  
> sum(x)  
[1] 185  
> mean(x)  
[1] 46.25  
> sd(x)  
[1] 39.44933  
>

[sum หาค่าผลรวม, mean หาค่าเฉลี่ย, sd หาค่าส่วนเบี่ยงเบนมาตรฐาน]

---

## Lesson 6: Create Our First Function

-เราสามารถสร้าง function ไว้ใช้งานเองได้ เช่น:



The screenshot shows the RStudio interface with two main panes. The left pane is the 'Code' editor containing R code. The right pane is the 'Console' showing the execution of that code.

```
1 x <- c(10, 25, 50, 100)
2
3 sum(x)
4 mean(x)
5 sd(x)
6
7 greeting <- function() {
8   print("Hello World!")
9 }
```

```
> greeting <- function() {
+   print("Hello World!")
+ }
> greeting
function() {
  print("Hello World!")
}
> greeting()
[1] "Hello World!"
>
```

[สร้าง function ชื่อ greeting ถ้าเรียกใช้แล้วจะแสดงผล Hello World! ออกบาก]

- ข้อดีของ function คือ reusable

-function สามารถรับ input ได้ เช่น:

The screenshot shows the RStudio interface. The top bar has tabs for 'Untitled1' and 'Console'. The main area is divided into two panes: a script editor on the left and a console on the right. The script editor contains the following R code:

```
1 sum(x)
2 mean(x)
3 sd(x)
4
5
6
7 greeting <- function() {
8   print("Hello World!")
9 }
10
11 greeting_name <- function(name) {
12   print(paste("Hello", name))
13 }
```

The console pane shows the output of running the script:

```
> greeting_name("Pooh")
[1] "Hello Pooh"
>
```

[สร้าง function ชื่อ greeting\_name ถ้าเรียกใช้แล้วจะแสดงผล Hello ตามด้วยชื่อที่กรอกไว้ในวงเล็บอookma]

-function สามารถซ้อน function ได้ เช่น:

The screenshot shows an RStudio interface with the following details:

- Code Editor:** The left pane displays R code. Lines 9 through 18 are visible, including a function definition for `greeting_name` and a call to it with the argument "Pooh".
- Console:** The right pane shows the R session output. It starts with the prompt `> func()`, followed by the result [1] "Hello World!". Then, after a new line, it shows the result of calling `greeting_name("Pooh")`, which is [1] "Hello Pooh".
- Status Bar:** At the bottom, the status bar indicates "R Script" and shows the file path "C:\Users\...".

[สร้าง function ชื่อ func เพื่อใช้งาน 2 function ข้างในปีกการร้องกับ]

# Lesson 7: Function Parameter & Argument

-ในแต่ละ function จะมี parameter และ argument \*ใช้ได้กับทุกภาษา

-parameter គឺជាដែលក្នុង function ត្រូវបានដាក់ជានាមីនុយ

-argument คือ value จริง ๆ ที่เราใส่ลงไปใน parameter ตอนเรารัน function เช่น:

```
greeting() <- function( name = "Pooh" ) { print(paste("Hi!", name)) }
```

[name เป็น parameter ส่วน “Pooh” เป็น argument]

-ถ้าเราใส่ default parameter ลงใน function แล้ว ตอนเราเรียกใช้งาน function ระบบจะเรียก default parameter ขึ้นมาด้วย แต่ เราสามารถ overwrite default parameter ได้ เช่น:

The screenshot shows an RStudio interface. On the left, the code editor contains the following R code:

```
7 greeting <- function() {  
8   print("Hello World!")  
9 }  
10  
11 greeting_name <- function(name = "Pooh") {  
12   print(paste("Hello", name))  
13 }  
14
```

On the right, the console window shows the following output:

```
> greeting_name()  
[1] "Hello Pooh"  
> greeting_name(name = "Mary")  
[1] "Hello Mary"  
> greeting_name("Mary")  
[1] "Hello Mary"  
>
```

-function สามารถรับได้มากกว่า 1 parameter เช่น:

The screenshot shows an RStudio interface. On the left, the code editor contains the following R code:

```
7 greeting <- function() {  
8   print("Hello World!")  
9 }  
10  
11 greeting_name <- function(name = "Pooh", age = 23) {  
12   print(paste("Hello", name))  
13   print(paste("Age =", age))  
14 }  
15
```

On the right, the console window shows the following output:

```
> greeting_name()  
[1] "Hello Pooh"  
[1] "Age = 23"  
> greeting_name(name = "Mary", age = 19)  
[1] "Hello Mary"  
[1] "Age = 19"  
> greeting_name(age = 19, name = "Mary")  
[1] "Hello Mary"  
[1] "Age = 19"
```

-ให้ระวังเรื่อง positional error จากการใส่ parameter คลบคล้ำกัน เช่น greeting\_name(19, "Mary")]

## Lesson 8: Function Kata

-เราจะฝึกเขียน function กั้งหนด 3 function ได้แก่:

1. add\_two\_nums() [function บวกเลข]
2. cube() [ยกกำลัง default คือ ยกกำลัง 3]
3. count\_ball()

-return() ไม่จำเป็นต้องเขียนบก็ได้ ได้ผลลัพธ์เท่ากัน

-add\_two\_nums():

```

Function_kata.R* ×
1 #add_two_nums()
2 add_two_nums <- function(value1, value2) {
3   value1 + value2
4 }

Console Terminal × Background Jobs ×
R 4.2.1 ~/→
> add_two_nums <- function(value1, value2) {
+   value1 + value2
+ }
> add_two_nums(10, 20)
[1] 30
>

```

-cube():

```

Function_kata.R* ×
1 #add_two_nums()
2 add_two_nums <- function(value1, value2) {
3   value1 + value2
4 }
5 cube <- function(base, power = 3) {
6   base ** power
7 }

Console Terminal × Background Jobs ×
R 4.2.1 ~/→
> cube(10)
[1] 1000
> cube(10, 2)
[1] 100
>

```

-count\_ball():

```

Function_kata.R* ×
5 cube <- function(base, power = 3) {
6   base ** power
7 }
8
9 balls <- c("red", "red", "blue", "green",
10           "green", "green", "green", "red")
11
12 count_ball <- function(balls, color) [
13   sum(balls == color)
14 ]

Console Terminal × Background Jobs ×
R 4.2.1 ~/→
> count_ball(balls, "red")
[1] 3
> count_ball(balls, "blue")
[1] 1
> count_ball(balls, "green")
[1] 4
>

```

## Lesson 9: Looping over a dataframe

-การ refactor คือการปรับรูปแบบการเขียนโค้ดให้อ่านง่ายขึ้น และแสดงผลเดียวกัน หรือรันไวขึ้น แต่ได้ผลลัพธ์เหมือนเดิม

-การดึง column จาก dataframe ใน R เขียนได้หลายแบบ เช่น เราต้องการดึง column ชื่อ col\_name ที่มี column index = 5:

1. data\$col\_name
2. data[["col\_name"]]
3. data[[5]]

-ใน R จะมี dataset ในตัว เราสามารถเรียกใช้งานได้ตามความเหมาะสม สามารถเช็ค dataset ได้ด้วย data()

-เราสามารถเช็คจำนวนของ row และ column ใน dataset ได้ด้วย nrow() และ ncol()

- เราสามารถใช้ for loop ในการดูชื่อของ column กั้งหมดใน dataset ได้ เช่น:

```
1 nrow(USArrests)
2 ncol(USArrests)
3 head(USArrests)
4
5 for (i in 1:ncol(USArrests)) {
6   print(names(USArrests)[i])
7 }
```

```
R 4.2.1 ~/r
California    9.0    276    91 40.6
Colorado      7.9    204    78 38.7
> for (i in 1:ncol(USArrests)) {
+   print(names(USArrests)[i])
+ }
[1] "Murder"
[1] "Assault"
[1] "UrbanPop"
[1] "Rape"
```

```
1 nrow(USArrests)
2 ncol(USArrests)
3 head(USArrests)
4
5 for (i in 1:ncol(USArrests)) {
6   print(names(USArrests)[i])
7   print(mean(USArrests[[i]]))
8 }
9
10 |
```

```
+ }
[1] "Murder"
[1] 7.788
[1] "Assault"
[1] 170.76
[1] "UrbanPop"
[1] 65.54
[1] "Rape"
[1] 21.232
>
```

- เราสามารถทำการคำนวณค่าใน column (Aggregate) ได้เหมือนใน Google Sheets เช่น:

```
# refactor our code for more readability
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))
  }
}

# test our code with mtcars
cal_mean_by_col(mtcars)
```

[หาค่าเฉลี่ยของค่าในแต่ละ column ของ database และแสดงผลชื่อของ column : ค่าเฉลี่ยของค่าใน column นั้น]

```
12
13 # refactor our code for more readability
14 cal_mean_by_col <- function(df) {
15   col_names <- names(df)
16
17   for (i in 1:ncol(df)) {
18     avg_col <- mean(df[[i]])
19     print(paste(col_names[i], ":", avg_col))
20   }
21 }
22
```

```
> # test our code with mtcars
> cal_mean_by_col(mtcars)
[1] "mpg : 20.090625"
[1] "cyl : 6.1875"
[1] "disp : 230.721875"
[1] "hp : 146.6875"
[1] "drat : 3.5965625"
[1] "wt : 3.21725"
[1] "qsec : 17.84875"
[1] "vs : 0.4375"
```

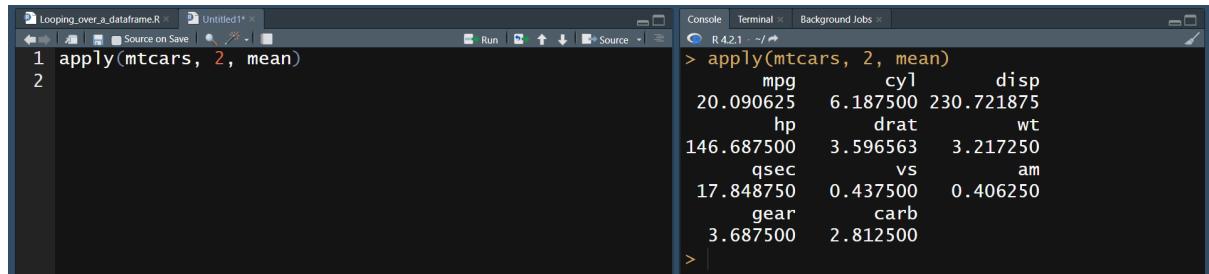
# Lesson 10: apply() coolest function

-อ่านเพิ่มเติมเกี่ยวกับ apply() ได้ที่:

<https://www.rdocumentation.org/packages/base/versions/3.6.2/topics/apply>.

-เราสามารถใช้ apply() เพื่อคืนค่า row หรือ column ที่ได้จากการ apply function กับ array หรือ matrix ได้ แม่นอนว่าสามารถใช้กับ database ได้ด้วย เช่น:

```
avg <- apply(dataset, MARGIN 2, mean)
```



The screenshot shows the RStudio interface with two panes. The left pane is the 'Source' tab containing the R code:

```
1 apply(mtcars, 2, mean)
```

The right pane is the 'Console' tab showing the output of the code execution:

```
> apply(mtcars, 2, mean)
   mpg      cyl      disp
20.090625 6.187500 230.721875
          hp      drat      wt
146.687500 3.596563  3.217250
          qsec      vs      am
17.848750  0.437500  0.406250
          gear      carb
            3.687500  2.812500
>
```

[apply function mean กับ dataset mtcars โดย apply mean กับทุก column ใน dataset]

\*MARGIN 1 คือวิ่งทุก row ส่วน MARGIN 2 คือวิ่งทุก column

-เราสามารถเปลี่ยน mean เป็น sum, sd, หรือ median เพื่อหาค่าทางสถิติอื่น ๆ ได้เช่นกัน และไม่จำกัดเพียง function ทางสถิติเหล่านี้ เราสามารถใช้ function อื่น ๆ กับ apply ได้ด้วย