

R 102 / Control Flow and Function

Control Flow

```
index.html
34 const events = [
35   'dragenter',
36   'dragleave',
37   'dragover', // to allow drop
38   'drop'
39 ];
40 events.forEach(e => {
41   fileDropZone.addEventListener(e, (ev) => {
42     ev.preventDefault();
43     if (ev.type === 'dragenter') {
44       fileDropZone.classList.add('solid-border');
45     }
46     if (ev.type === 'dragleave') {
47       fileDropZone.classList.remove('solid-border');
48     }
49     if(ev.type === 'drop') {
50       fileDropZone.classList.remove('solid-border');
51       handleFiles(ev.dataTransfer.files)
52         .then(values => values.map(tag => {
53           tag.setAttribute('class', 'border rounded-lg previous');
54           fileDropZone.appendChild(tag);
55         }));
56     }
57   });
58 });
59 
```

Control Flow คือหนึ่งใน building blocks ที่สำคัญของการเขียนโปรแกรม (ความรู้ในบทเรียนนี้ใช้ได้กับทุกภาษาและเครื่อง) ใน R เราไม่ control flow สำคัญอยู่สามตัวคือ

1. `if`
2. `for`
3. `while`

Note - เวลาเราพิมพ์ keyword `if` `for` `while` ใน script จะมี highlight syntax ให้เราด้วย

💡 หน้ากี๊ของ **Control Flow** คือการควบคุมพฤติกรรมของโปรแกรมที่เราเขียน ตัวอย่างเช่น

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

ถ้าคะแนนสอบมากกว่าหรือเท่ากับ 80 คะแนน สอบผ่าน "passed" แต่ถ้าคะแนนไม่ถึงเกณฑ์คือสอบตก "failed"

เราเขียน `if` เพื่อกำหนดเส้นทาง (path) การทำงานของโปรแกรม

หรือถ้าเราเขียน `for` เราจะสามารถ loop ตัวแปรหรือ object ที่เราประกาศไว้ใน R ได้ เช่น การตัดเกรดคะแนนนักเรียน 5 คนในตัวแปร `scores`

```

for_loop_example.R
1 # for loop example
2 scores <- c(50, 88, 75, 80, 95)
3
4 for (score in scores) {
5   if (score >= 80) {
6     print("passed")
7   } else {
8     print("failed")
9   }
10 }

Console Terminal Background Jobs
R 4.2.1 · /cloud/project/ ↗
> # for loop example
> scores <- c(50, 88, 75, 80, 95)
>
> for (score in scores) {
+   if (score >= 80) {
+     print("passed")
+   } else {
+     print("failed")
+   }
+ }
[1] "failed"
[1] "passed"
[1] "failed"
[1] "passed"
[1] "passed"
> |

```

💡 ในบทเรียนนี้เราจะเรียนวิธีการเขียน control flow และ functions ด้วยกันนะครับ มีกั้งหมด 10 วิดีโอ ใช้เวลาเรียนประมาณหนึ่งชั่วโมง

IF

การเขียนเงื่อนไข **IF**

เป็นพื้นฐานของ computer program ทุกภาษาจะมีโครงสร้างการเขียนเหมือนกันเลย คือ condition, if TRUE, else FALSE

```

T.R*
1 ## control flow
2 ## =IT() in google sheets
3
4 score <- 95
5
6 if (score >= 90) {
7   print("Passed")
8 } else {
9   print("Failed")
10 }
11
12 if (score >= 90) {
13   print("Passed")
14 } else if (score >= 50) {
15   print("ok")
16 } else {
17   print("Enroll again!")
18 }
19

Console Terminal Background Jobs
R 4.2.2 · ~/ ↗
> if (score >= 90) {
+   print("Passed")
+ } else if (score >= 50) {
+   print("ok")
+ } else {
+   print("Enroll again!")
+ }
[1] "Passed"
> if (score >= 90) {
+   print("Passed")
+ } else {
+   print("Failed")
+ }
[1] "Passed"
> |

```

หากคะแนนสอบมากกว่า 90 ให้ขึ้น Passed

```

score <- 95

if (score >= 90) {

```

```

    print("Passed")
} else {
    print("Failed")
}

```

ເພີ່ມເຈື້ອນໄຂວິນໆ

```

if (score >= 90) {
    print("Passed")
} else if (score >= 50) {
    print("ok")
} else {
    print("Enroll again!")
}

```

ifelse

`ifelse()` syntax ເຊິ່ງເຫັນເມື່ອນກັບ `=IF()` ໃນ Google Sheets/ Excel

`ifelse(condition, TRUE, FALSE)`

The screenshot shows the RStudio interface with two panes. The left pane is the 'Code' editor with the following R code:

```

# if
# =IF() in google sheets
score <- 38
if (score >= 90) {
  print("Passed")
} else {
  print("Failed")
}
if (score >= 90) {
  print("Passed")
} else if (score >= 50) {
  print("Ok")
} else {
  print("Enroll again!")
}

```

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

```

> ifelse(score >= 80, "Passed", "Failed")
[1] "Failed"
>
> score <- 95
> ifelse(score >= 80, "Passed", "Failed")
[1] "Passed"
> score <- 72
> ifelse(score >= 80, "Passed", "Failed")
[1] "Failed"
>
> ifelse(score >= 90, "Passed", ifelse(
+   score >= 50, "OK", "Enroll Again"
+ ))
[1] "OK"
>
> score <- 92
> ifelse(score >= 90, "Passed", ifelse(
+   score >= 50, "OK", "Enroll Again"
+ ))
[1] "Passed"
>

```

FOR

loop ใน R ส่วนตัวแอดดิชันน้อยมาก เพราะ code ใน R รันแบบ **vectorization** อยู่แล้ว และในวีดีโอ sprint ต่อๆไป แอดจะมีสอนใช้

apply()

family ด้วย ตอนวีเคราะห์ข้อมูลจริงเราแทบจะไม่ต้องเขียน manual

for

loop เองเลยครับใน R

The screenshot shows the RStudio interface with a script file named "Untitled1.R" open. The code in the script is:

```
20 # for
21 friends <- c("Toy", "John", "Mary", "Anna")
22
23 for (friend in friends) {
24   print(paste("Hi!", friend))
25 }
26
27 # ve
```

In the RStudio interface, there are tabs for "Console", "Terminal", and "Background Jobs". The "Console" tab is active, showing the output of the script:

```
R 4.2.1 - /cloud/project/
> for (friend in friends) {
+   print(friend)
+ }
[1] "Toy"
[1] "John"
[1] "Mary"
[1] "Anna"
> for (friend in friends) {
+   print(paste("Hi!", friend))
+ }
[1] "Hi! Toy"
[1] "Hi! John"
[1] "Hi! Mary"
[1] "Hi! Anna"
```

ใช้ Vectorization

Function paste

The screenshot shows the RStudio interface with a script file named "T.R" open. The code in the script is:

```
1 friends <- c("Toy", "John", "Mary", "Anna", "David")
2 paste("Hi!", friends)
```

In the RStudio interface, there are tabs for "Console", "Terminal", and "Background Jobs". The "Console" tab is active, showing the output of the script:

```
R 4.2.2 - ~/
> paste("Hi!", friends)
[1] "Hi! Toy"  "Hi! John" "Hi! Mary" "Hi! Anna" "Hi! David"
```

```
friends <- c ("Toy", "John", "Mary", "Anna", "David")
paste("Hi!" , friends)
```

สามารถใช้กับการบวกลบได้

```

File Edit Code View Plots Session Build Debug Profile Tools Help
Untitled1 * Go to file/function Addins R 4.2.1
20 # for
21 friends <- c("Toy", "John", "Mary", "Anna", "David")
22
23 for (friend in friends) {
24   print( paste("Hi!", friend) )
25 }
26
27 # vectorization
28
29 paste("Hi!", friends)
30
31 nums <- c(5, 10, 12, 20, 25)
32 nums <- nums + 2
33
34
35
36
37
38
39
40
41

```

Console Terminal Background Jobs
R 4.2.1 - /cloud/project/
> nums
[1] 5 10 12 20 25
> nums + 2
[1] 7 12 14 22 27
>

และ add sign ไปที่ตัวบันเลขเพื่อ update ค่า

```

File Edit Code View Plots Session Build Debug Profile Tools Help
Untitled1 * Go to file/function Addins R 4.2.1
20 # for
21 friends <- c("Toy", "John", "Mary", "Anna", "David")
22
23 for (friend in friends) {
24   print( paste("Hi!", friend) )
25 }
26
27 # vectorization
28
29 paste("Hi!", friends)
30
31 nums <- c(5, 10, 12, 20, 25)
32 nums <- nums + 2
33
34
35
36
37
38
39
40
41

```

Console Terminal Background Jobs
R 4.2.1 - /cloud/project/
> nums
[1] 7 12 14 22 27
>

หากใส่ เข้าไปด้วยจะดำเนินการ update ค่าและ print ออกมาให้เลย

The screenshot shows the RStudio interface. In the top-left pane, there is an R script named "Untitled1.R". The code contains several lines of R code, including loops and vectorization examples. In the bottom-left pane, there is a "File Script" tab. On the right side of the interface, the "Console" tab is active, showing the output of the R code. The console output shows the results of various operations, such as printing vectors and performing arithmetic operations like subtraction by 2.

```
20 # for
21 friends <- c("Toy", "John", "Mary", "Anna", "David")
22
23 for (friend in friends) {
24   print(paste("Hi!", friend))
25 }
26
27 # vectorization
28
29 paste("Hi!", friends)
30
31 nums <- c(5, 10, 12, 20, 25)
32 nums <- nums + 2
33
34 for (num in nums) {
35   print(num - 2)
36 }
37
38 (nums <- nums - 2)
```

```
R 4.2.1 · /cloud/project/
> (nums <- nums - 2)
[1] 5 10 12 20 25
> nums
[1] 5 10 12 20 25
>
```

WHILE

อย่าลืมอพเดตตัวแปร `count` ก่อนใช้ใน `while` loop ไม่งั้นเราจะติดใน `infinite loop`

ตัวอย่าง

จะเห็นว่า Query ออกมา loop ไม่มีวันจบ

The screenshot shows the RStudio interface. In the top-left pane, there is an R script named "Untitled1.R". The code contains a simple `while` loop that prints "Hi!" five times. In the bottom-left pane, there is a "File Script" tab. On the right side of the interface, the "Console" tab is active, showing the output of the R code. The console output shows the word "Hi!" repeated 15 times, indicating that the loop is running indefinitely.

```
40 # while loop
41 count <- 0
42
43 while (count < 5) {
44   print("Hi!")
45 }
```

```
R 4.2.1 · /cloud/project/
[1] "Hi!"
[1] "Hi!
...."
```

```
count <- 0

while (count < 5) {
  print("Hi!")
}
```

แก้ Code เพื่อให้วงแคร์ 5 รอบ

The screenshot shows the RStudio interface. The code editor on the left contains the following R code:

```
1 count <- 0
2
3 while (count < 5) {
4   print("Hi!")
5   count <- count +1
6 }
7
```

The console window on the right shows the output of the code execution:

```
R 4.2.2 : ~/ -->
> count <- 0
> while (count < 5) {
+   print("Hi!")
+   count <- count +1
+
[1] "Hi!"
[1] "Hi!"
[1] "Hi!"
[1] "Hi!"
[1] "Hi!"
> |
```

```
count <- 0

while (count < 5) {
  print("Hi!")
  count <- count +1
}
```

Function

- sum = คำนวณยอด
- mean = หาค่าเฉลี่ย
- sd = standard deviation

The screenshot shows the RStudio interface. On the left, a script editor window titled 'T.R* x' contains the following R code:

```
## function
## input -> f() -> output

x <- c(10, 25, 50, 100)

sum(x)
mean(x)
sd(x)
```

On the right, the 'Console' tab of the R session window shows the execution of the code:

```
> x <- c(10, 25, 50, 100)
> sum(x)
[1] 185
> mean(x)
[1] 46.25
> sd(x)
[1] 39.44933
>
```

```
## function
## input -> f() -> output

x <- c(10, 25, 50, 100)

sum(x)
mean(x)
sd(x)
```

Create Our First Function

สร้าง Function ใช้ keyword

R ใช้ keyword `function` ในการประกาศ function ใหม่

The screenshot shows the RStudio interface. On the left, a script editor window titled 'T.R* x' contains the following R code:

```
# create our first function
greeting <- function() {
  print("Hello World!")
}

greeting_name <- function(name) {
  print( paste("Hello!", name))
}

func <- function() {
  greeting()
  greeting_name("Toy")
}
```

On the right, the 'Console' tab of the R session window shows the execution of the code:

```
> greeting <- function() {
+   print("Hello World!")
+ }
> greeting_name <- function(name) {
+   print( paste("Hello!", name))
+ }
> func <- function() {
+   greeting()
+   greeting_name("Toy")
+ }
> greeting()
[1] "Hello World!"
> greeting
function() {
  print("Hello World!")
}
> greeting_name()
```

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

```
58 # create our first function
59
60 greeting <- function() {
61   print("Hello World!")
62 }
63
64 greeting_name <- function(name) {
65   print( paste("Hello!", name) )
66 }
67
68
69
70
```

The right pane shows the R console output:

```
> greeting_name("Toy")
[1] "Hello! Toy"
> greeting_name("John")
[1] "Hello! John"
> greeting_name("Anna")
[1] "Hello! Anna"
>
```

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

```
60 greeting <- function() {
61   print("Hello World!")
62 }
63
64 greeting_name <- function(name) {
65   print( paste("Hello!", name) )
66 }
67
68 func <- function() {
69   greeting()
70   greeting_name("Toy")
71 }
72
```

A yellow circle highlights the character 'I' in the line 'func <- function()'.

The right pane shows the R console output:

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

```
# create our first function

greenting <- function() {
  print("Hello World!")
}

greetting_name <- function(name) {
  print( paste("Hello!", name))
}

func <- function() {
  greenting()
  greetting_name("Toy")
}
```

Function Parameter & Argument

คำศัพท์ที่เราใช้ในการเขียน `function` มือย่อส่องคำ (concept นี้ใช้ได้กับทุกภาษาเลย)

- `parameter`

- argument

Parameter คือชื่อ named input ที่เราใส่ใน function ส่วน **Argument** คือ actual value ที่เราใส่ไปใน parameter นั้นๆตอนเรารัน function

ตัวอย่างการเขียน parameter และ argument ใน R

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

- name คือ parameter
- "Toy" คือ argument

The screenshot shows the RStudio interface. On the left, the code editor window titled 'intro_r.R*' contains the following R code:

```

8 # create our first function
9
10 greeting <- function() {
11   print("Hello World!")
12 }
13
14 # name, age => parameter
15 # "Toy", 25 => argument
16 greeting_name <- function(name = "Toy", age = 25) {
17   print( paste("Hello!", name) )
18   print( paste("Age:", age) )
19 }
20
21 func <- function() {
22   greeting()
23   greeting_name("Toy")
24 }
25
26
27
28

```

On the right, the 'Console' tab shows the execution of the code and its output:

```

> greeting_name()
[1] "Hello! Toy"
[1] "Age: 25"
>
> greeting_name(name="David", age=18)
[1] "Hello! David"
[1] "Age: 18"
>
> greeting_name(age=18, name="David")
[1] "Hello! David"
[1] "Age: 18"
>
> greeting_name(18, "David")
[1] "Hello! 18"
[1] "Age: David"
>
> greeting_name("David", 18)
[1] "Hello! David"
[1] "Age: 18"
>

```

```
# create our first function

greeting <- function() {
  print("Hello World!")
}

greeting_name <- function(name = "Toy", age = 25) {
  print( paste("Hello!", name) )
  print( paste("Age:", age) )
}

func <- function() {
  greeting()
  greeting_name("Toy")
}
```

Function Kata

Practice Practice Practice

Ψ มาลองฝึกเขียน 3 functions ใน R ด้วยกันครับ

- `add_two_nums()` = + - *
- `cube()` = ยกกำลัง
- `count_ball()` = นับจำนวน

```
# add_two_nums() function
add_two_nums <- function(val1, val2) {
  return(val1 + val2)
}

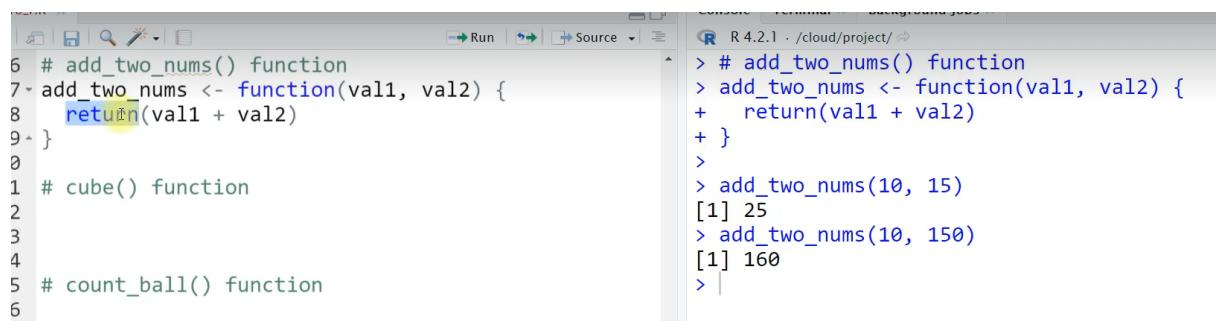
# cube() function
cube <- function(base, power=3) {
  return(base ** power)
}

# count_ball() function
balls <- c("red", "red", "blue", "green",
          "green", "green", "green", "red")

count_ball <- function(balls) {
  sum(balls == "red")
}
```

add_two_nums

หรือไม่ต้องเขียน return ก็ได้



The screenshot shows the RStudio interface. On the left, the code editor displays the three functions: `add_two_nums()`, `cube()`, and `count_ball()`. The `add_two_nums()` function has its `return` statement highlighted in yellow. On the right, the R console window shows the following session:

```
R 4.2.1 · /cloud/project/
> # add_two_nums() function
> add_two_nums <- function(val1, val2) {
+   return(val1 + val2)
+ }
>
> add_two_nums(10, 15)
[1] 25
> add_two_nums(10, 150)
[1] 160
>
```

cube

เลขยกกำลัง

The screenshot shows the RStudio interface. On the left, the code editor displays `intro_r.R` with the following content:

```
76 # add_two_nums() function
77 add_two_nums <- function(val1, val2) {
78   val1 + val2
79 }
80
81 # cube() function
82 cube <- function(base, power=3) {
83   return(base ** power)
84 }
85
86 # count_ball() function
87
88
89
```

On the right, the console window shows the execution of the code:

```
> cube(5)
[1] 125
> cube(5, power=4)
[1] 625
> cube(5, 4)
[1] 625
> cube(power=4, base=5) # 5**4
[1] 625
>
```

Count_ball

นับจำนวนค่าที่เป็น character

The screenshot shows the RStudio interface. On the left, the code editor displays `intro_r.R` with the following content:

```
1 # cube() function
2 cube <- function(base, power=3) {
3   return(base ** power)
4 }
5
5 # count_ball() function
7 balls <- c("red", "red", "blue", "green",
8           "green", "green", "green", "red")
9
9 count_ball <- function(balls) {
10   sum(balls == "red")
```

On the right, the console window shows the execution of the code:

```
> balls
[1] "red"    "red"    "blue"   "green"  "green"
[6] "green"  "green"  "red"
> balls == "red"
[1] TRUE TRUE FALSE FALSE FALSE FALSE FALSE
[8] TRUE
> sum(balls == "red")
[1] 3
> sum(balls == "blue")
[1] 1
> sum(balls == "green")
[1] 4
>
```

เพิ่มเงื่อนไข Count จำนวนของสีบลลแต่ละลูก

The screenshot shows the RStudio interface. On the left, the code editor displays `intro_r.R` with the following content:

```
81 # cube() function
82 cube <- function(base, power=3) {
83   return(base ** power)
84 }
85
86 # count_ball() function
87 balls <- c("red", "red", "blue", "green",
88           "green", "green", "green", "red")
89
90 count_ball <- function(balls, color) {
91   sum(balls == color)}
```

On the right, the console window shows the execution of the code:

```
> count_ball <- function(balls, color) {
+   sum(balls == color)
+ }
>
> count_ball(balls, "red")
[1] 3
> count_ball(balls, "green")
[1] 4
> count_ball(balls, "blue")
[1] 1
>
```

```

# add_two_nums() function
add_two_nums <- function(val1, val2) {
  return(val1 + val2)
}

# cube() function
cube <- function(base, power=3) {
  return(base ** power)
}

# count_ball() function
balls <- c("red", "red", "blue", "green",
          "green", "green", "green", "red")

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

```

Looping over a dataframe

Refactoring Your Code

💡 Refactor คือการปรับรูปแบบการเขียนโค้ดของเรา (re-structuring code) ให้อ่านง่ายขึ้น และงดผลลัพธ์ที่ไม่ต้องการ หรือรันได้เร็วขึ้น แต่ผลลัพธ์ยังคงมาเหมือนเดิม

Function `cal_mean_by_col()` ที่แอดเขียนในวิดีโอแสดงผลไม่ค่อยสวยงามเท่าไหร่ แอดเลย `refactor` ใหม่ ให้อ่านง่ายขึ้นด้วยโค้ดด้านล่าง

Tip - เวลาเรอやりจะดึงคอลัมน์จาก dataframe ใน R สามารถเขียนได้หลายแบบ สมมติเราอยากจะดึงคอลัมน์ชื่อ `"col_name"`, column index = 5 ใน dataframe ตัวอย่าง syntax ทั้งสามแบบนี้ได้ผลลัพธ์เหมือนกันเลย

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

JAVASCRIPT

```

# refactor our code for more readability
cal_mean_by_col <- function(df) {
  col_names <- names(df)

```

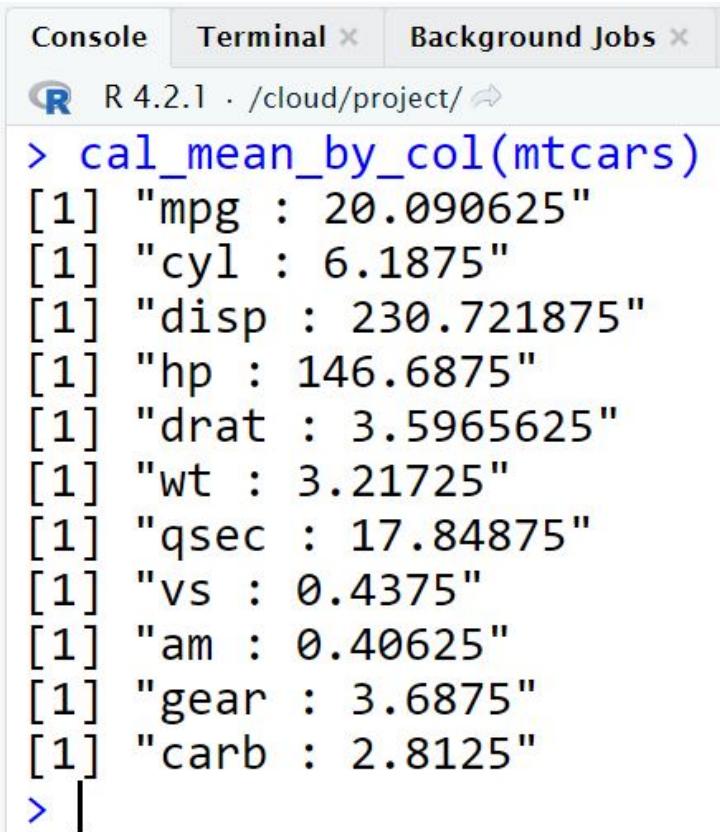
```

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)

```

Output ที่ได้จาก `cal_mean_by_col()` ที่เรา refactor ใหม่ ดุจดังขึ้นเยอะเลย 😊



The screenshot shows the RStudio interface with the 'Console' tab selected. The session starts with 'R 4.2.1 · /cloud/project/'. The command `> cal_mean_by_col(mtcars)` is run, followed by its output:

```

> 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"
[1] "am : 0.40625"
[1] "gear : 3.6875"
[1] "carb : 2.8125"
>

```

ตัวอย่างข้อมูลจาก database

```

data()

nrow(USArrests)
ncol(USArrests)
head(USArrests)

```

```

for (i in 1:ncol(USArrests)) {
  print( names(USArrests)[i] )
  print( mean(USArrests[[i]]) )
}

```

☰ Bootcamp 06 / Intro to R Programming

The screenshot shows the RStudio interface. On the left, the code editor displays a script named `intro_r.R` with the following content:

```

93
94 # loop over a dataframe
95 data()
96
97 nrow(USArrests)
98 ncol(USArrests)
99 head(USArrests)
100
101 cal_mean_by_col <- function(df) {
102   for (i in 1:ncol(df)) {
103     print( names(df)[i] )
104     print( mean(df[[i]]) )
105   }
106 }
107
108
109
110

```

On the right, the `Console` tab shows the output of the command `> cal_mean_by_col(USArrests)`:

```

> cal_mean_by_col(USArrests)
[1] "Murder"
[1] 7.788
[1] "Assault"
[1] 170.76
[1] "UrbanPop"
[1] 65.54
[1] "Rape"
[1] 21.232
>

```

ตัวอย่างที่ 2 Loop ข้อมูลที่เป็น mtcars

```
cal_mean_by_col(mtcars)
```

The screenshot shows the RStudio interface. On the left, the code editor displays a script named `intro_r.R` with the following content:

```

File Edit Code View Plots Session Build Debug Profile Tools Help
Go to file/function Addins
intro_r.R R data sets mtcars USArrests
Filter
mpg cyl disp hp drat wt qsec vs am gear carb
Lazda RX4 21.0 6 160.0 110 3.90 2.620 16.46 0 1 4 4
RX4 Wag 21.0 6 160.0 110 3.90 2.875 17.02 0 1 4 4
atsun 710 22.8 4 108.0 93 3.85 2.320 18.61 1 1 4 1
et 4 Drive 21.4 6 258.0 110 3.08 3.215 19.44 1 0 3 1
portabout 18.7 8 360.0 175 3.15 3.440 17.02 0 0 3 2
  Valiant 18.1 6 225.0 105 2.76 3.460 20.22 1 0 3 1
uster 360 14.3 8 360.0 245 3.21 3.570 15.84 0 0 3 4
terc 240D 24.4 4 146.7 62 3.69 3.190 20.00 1 0 4 2
Merc 230 22.8 4 140.8 95 3.92 3.150 22.90 1 0 4 2
Merc 280 19.2 6 167.6 123 3.92 3.440 18.30 1 0 4 4
Merc 280C 17.8 6 167.6 123 3.92 3.440 18.90 1 0 4 4
erc 450SE 16.4 8 275.8 180 3.07 4.070 17.40 0 0 3 3

```

On the right, the `Console` tab shows the output of the command `> cal_mean_by_col(mtcars)`:

```

> View(mtcars)
>
> cal_mean_by_col(mtcars)
[1] "mpg"
[1] 20.09062
[1] "cyl"
[1] 6.1875
[1] "disp"
[1] 230.7219
[1] "hp"
[1] 146.6875
[1] "drat"
[1] 3.596563
[1] "wt"
[1] 3.21725

```

apply() coolest function

ใน R มีฟังก์ชัน `apply()` กี่เราใช้แทนการเขียน `loop` ได้ก็ columns, rows สะดวกมาก
อ่านเพิ่มเติมเกี่ยวกับ `apply()` ได้ที่ <https://www.rdocumentation.org/packages/base/versions/3.6.2/topics/apply>

```
# apply function
avg_by_row_mtcars <- apply(mtcars, MARGIN=1, mean)

apply(mtcars, MARGIN=2 ,sum)
apply(mtcars, MARGIN=2 ,sd)
apply(mtcars, MARGIN=2 ,median)
```

The screenshot shows the RStudio interface. In the top-left, there's a file named 'ro_r.R' with a 'mtcars' tab open. Below it is a code editor with the following R script:

```
7
8 cal_mean_by_col <- function(df) {
9   col_names <- names(df)
0
1   for (i in 1:ncol(df)) {
2     avg_col <- mean(df[[i]])
3     print(paste(col_names[i], ":", avg_col))
4   }
5 }
6
7 # apply function
8 avg_by_row_mtcars <- apply(mtcars, MARGIN=1, mean) # by
9
0 apply(mtcars, MARGIN=2, sum)
1 apply(mtcars, MARGIN=2, sd)
2 apply(mtcars, MARGIN=2, median)
3
4
5
6
7
8
```

In the top-right, there's a 'Console' tab showing the output of the script. It includes the output of the custom function 'cal_mean_by_col' which prints the mean of each column, and the results of applying various functions (mean, sum, sd, median) across different dimensions of the mtcars dataset.

Function	MARGIN	Output
mean	1	avg_by_row_mtcars
sum	2	apply(mtcars, MARGIN=2, sum)
sd	2	apply(mtcars, MARGIN=2, sd)
median	2	apply(mtcars, MARGIN=2, median)

Reading - Hands On Programming with R

Reading Assignment

O'REILLY®



Hands-On Programming with R

WRITE YOUR OWN FUNCTIONS AND SIMULATIONS

Garrett Grolemund
Foreword by Hadley Wickham

บัคเรียนกี่เรียนจบ [Control Flow and Function](#) แล้ว แอดมีการบ้านให้อ่านหนังสือ Hands-On Programming with R บทที่ 1-7

Ψ <https://rstudio-education.github.io/hopr/>

เสร็จแล้วเขียนสรุปใน [Notion](#) ของตัวเอง แชร์ได้ที่ห้อง #r ใน discord โรงเรียนเรา南北

Not all readers are leaders, but all leaders are readers. - Harry S. Truman