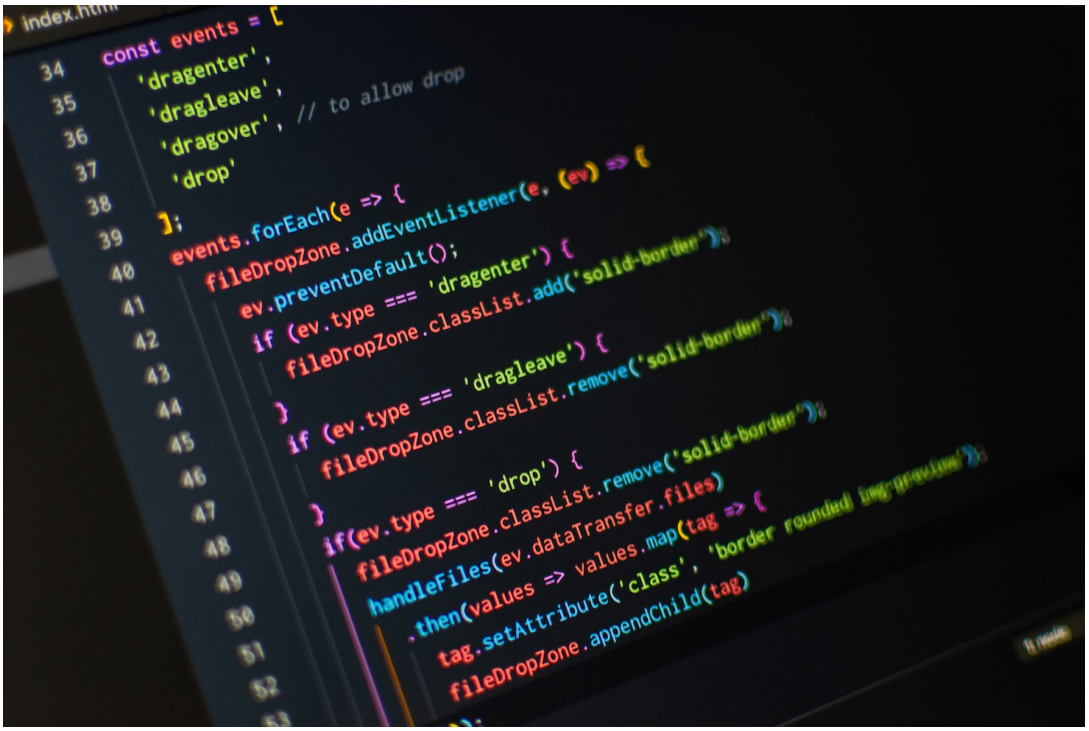
R 102 / Control Flow and Function

Control Flow



Control Flow คือหนึ่งใน building blocks ที่สำคัญของการเขียนโปรแกรม (ความรู้ในบทเรียนนี้ใช้ได้กับทุกภาษาเลยนะครับ) ใน R เรามี control flow สำคัญอยู่สามตัวคือ

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

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

🌵 หน้าที่ของ **Control Flow** คือการควบคุมพฤติกรรมของโปรแกรมที่เราเขียน ตัวอย่างเช่น

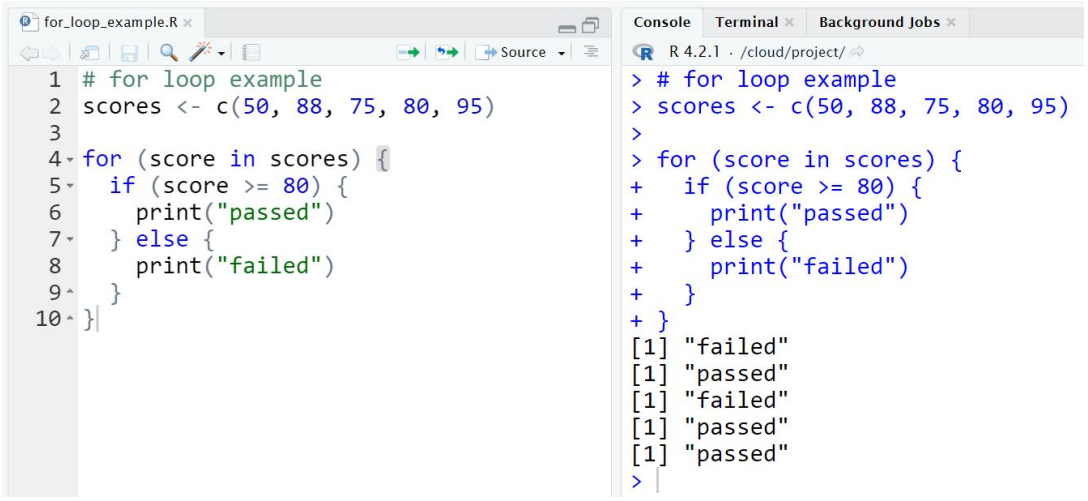
```
score <- 85
```

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

ถ้าคะแนนสอบมากกว่าหรือเท่ากับ 80 คะแนน สอบผ่าน "passed" แต่ถ้าคะแนนไม่ถึงเกณฑ์คือสอบตก "failed"

เราเขียน `if` เพื่อกำหนดเส้นทาง (path) การทำงานของโปรแกรม

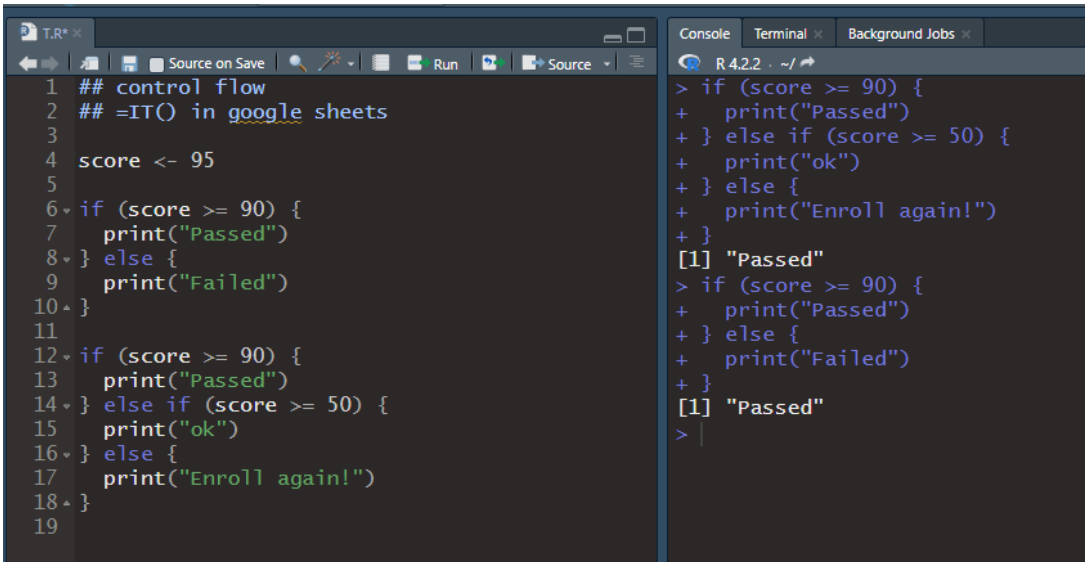
หรือถ้าเราเขียน `for` เราจะสามารถ loop ตัวแปรหรือ object ที่เราประกาศไว้ใน R ได้ เช่น การตัดเกรดคะแนนนักเรียน 5 คนในตัวแปร `scores`



🍌 ในบทเรียนนี้เราจะมาเรียนวิธีการเขียน control flow และ functions ด้วยกันนะครับ มีทั้งหมด 10 วีดีโอ ใช้เวลาเรียนประมาณหนึ่งชั่วโมง

IF

การเขียนเงื่อนไข `IF`
เป็นพื้นฐานของ computer program ทุกภาษาจะมีโครงสร้างการเขียนเหมือนกันเลย คือ condition, if TRUE, else FALSE



หากคะแนนสอบมากกว่า 90 ให้ขึ้น Passed

```
score <- 95

if (score >= 90) {
```

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

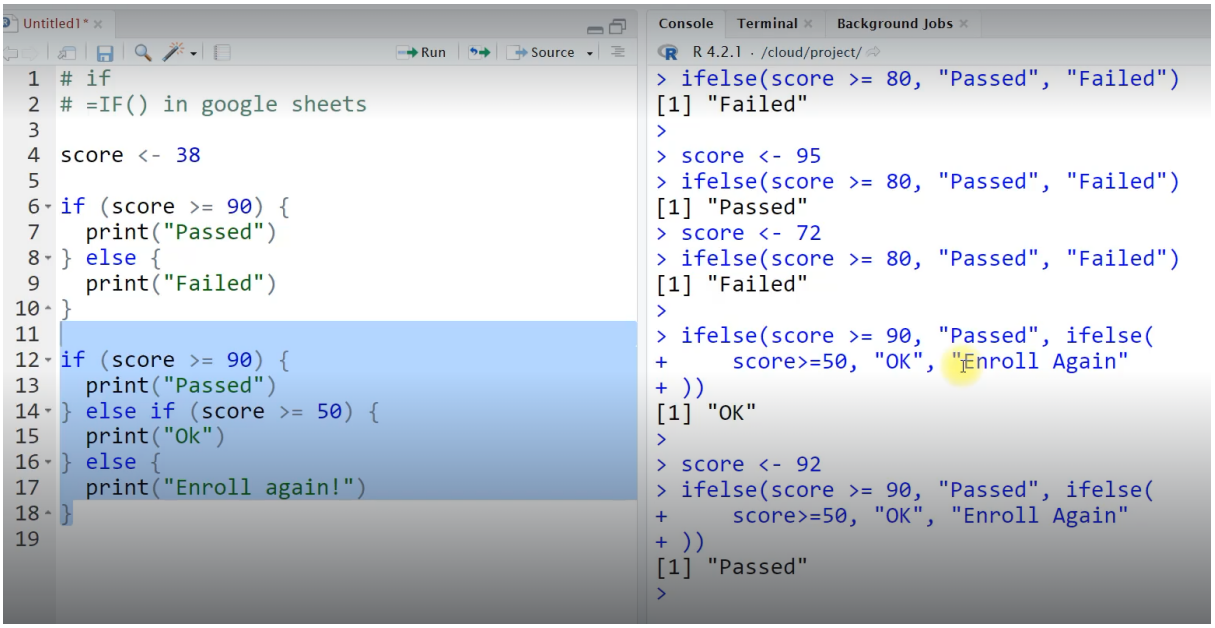
เพิ่มเงื่อนไขอื่นๆ

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

ifelse

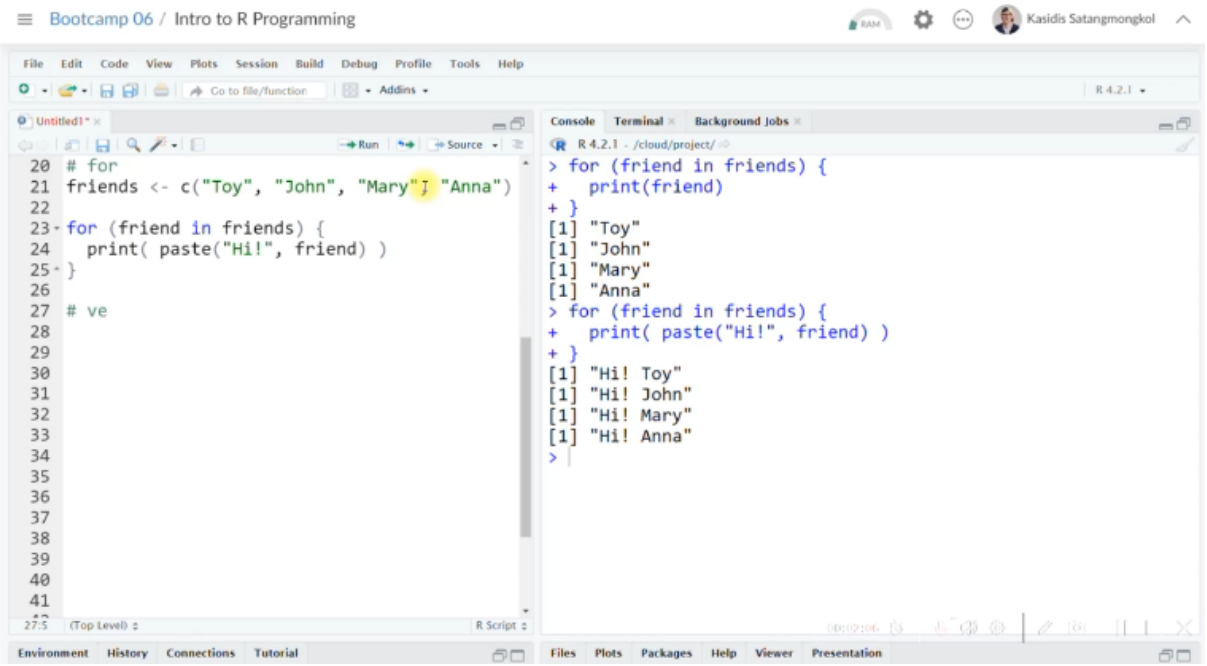
`ifelse()` syntax เขียนเหมือนกับ `=IF()` ใน Google Sheets/ Excel

`ifelse(condition, TRUE, FALSE)`



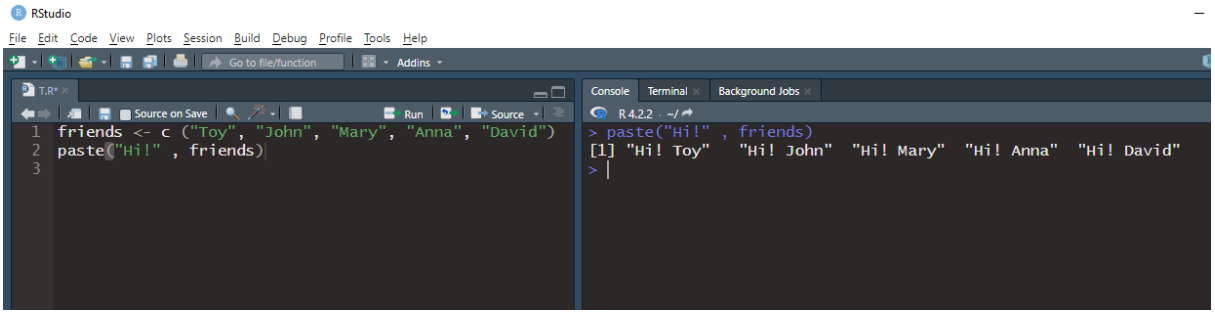
FOR

loop ใน R ส่วนตัวแอดใช้น้อยมาก เพราะ code ใน R รันแบบ `vectorization` อยู่แล้ว และในวีดีโอ sprint ต่อๆไป แอดจะมีสอนใช้ `apply()` family ด้วย ตอนวิเคราะห์ข้อมูลจริงเราแทบจะไม่ต้องเขียน manual `for` loop เองเลยครับใน R



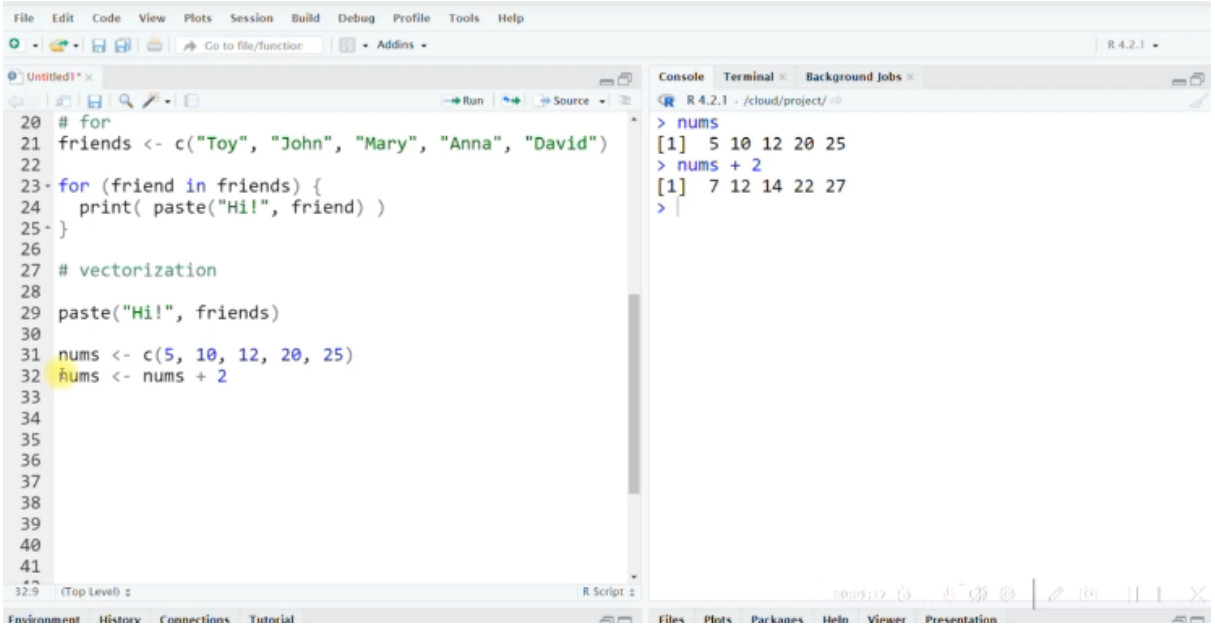
ใช้ Vectorization

Function paste

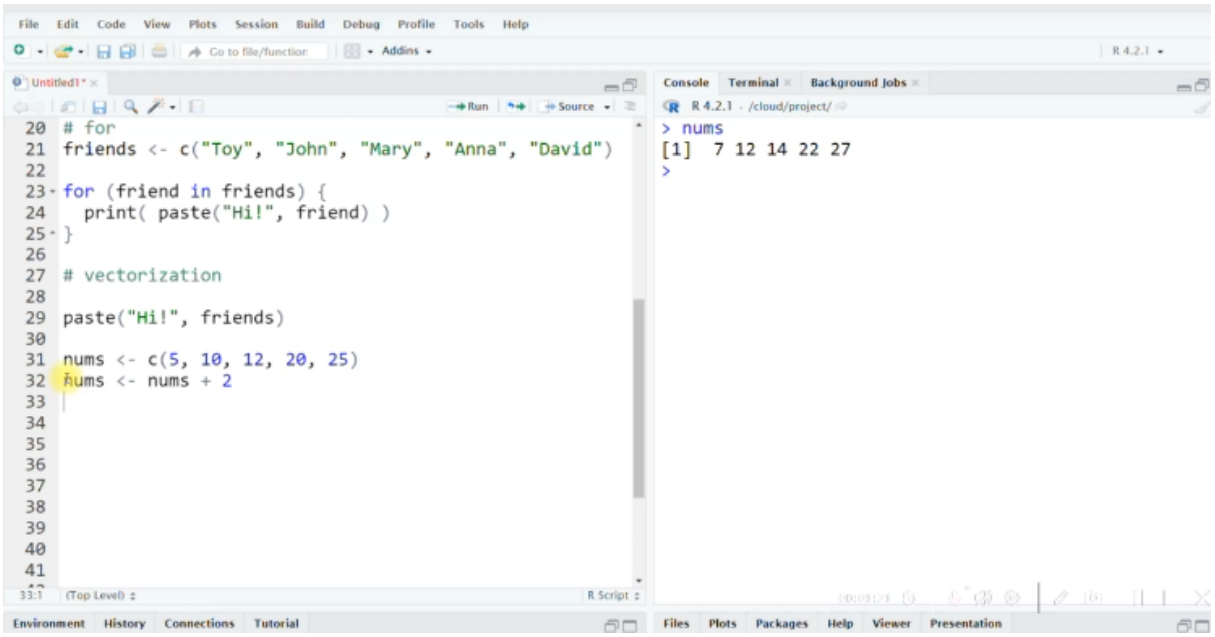



```
friends <- c ("Toy", "John", "Mary", "Anna", "David")
paste("Hi!" , friends)
```

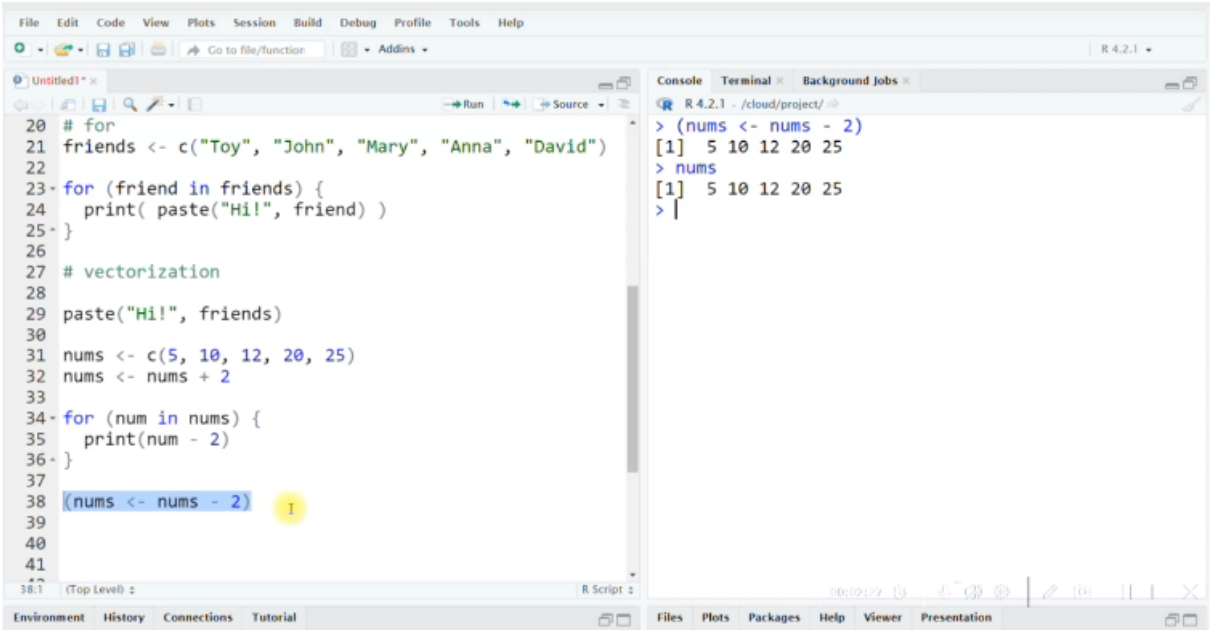
สามารถใช้กับการบวกลบได้



และ add sign ไปที่ตัวมันเองเพื่อ update ค่า



หากใส่ `()` เข้าไปด้วยจะดำเนินการ update ค่าและ print ออกมาให้เลย

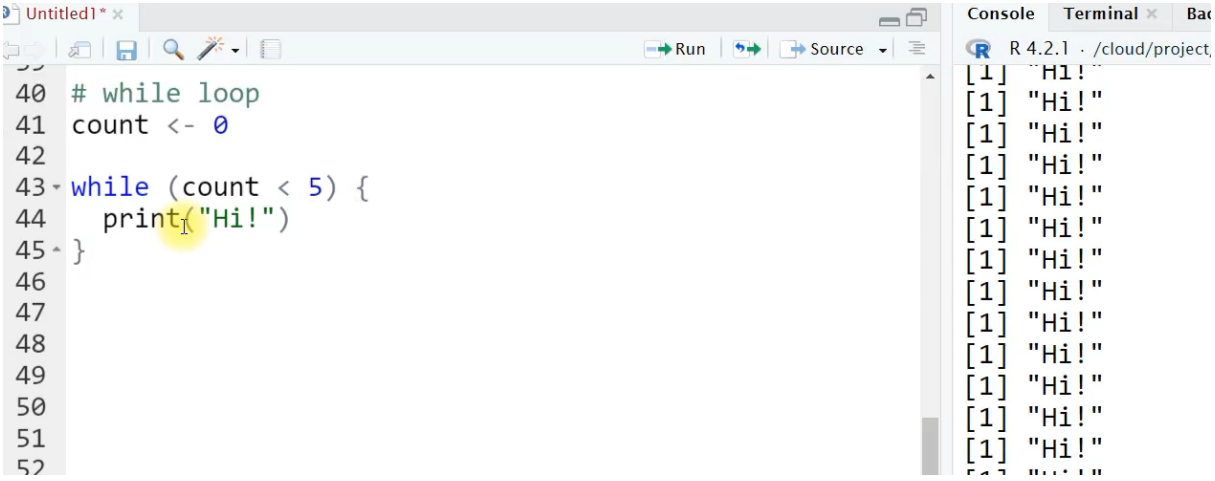


WHILE

อย่าลืมอัพเดทตัวแปร `count` ที่เราใช้ใน `while` loop ไม่งั้นเราจะติดใน `infinite loop`

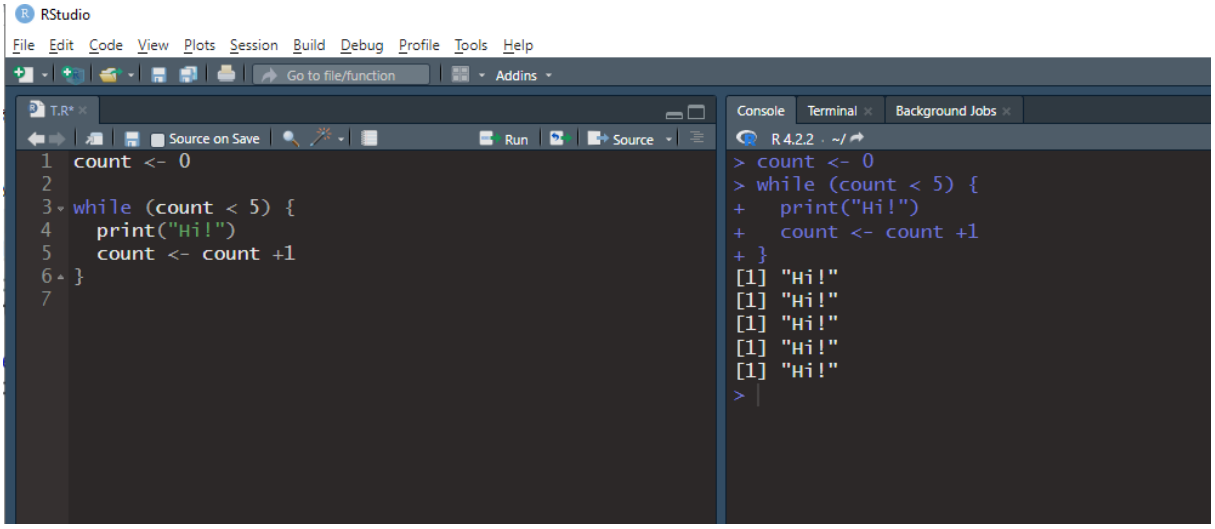
ตัวอย่าง

จะเห็นว่า Query ออกมา loop ไม่มีวันจบ




```
count <- 0

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

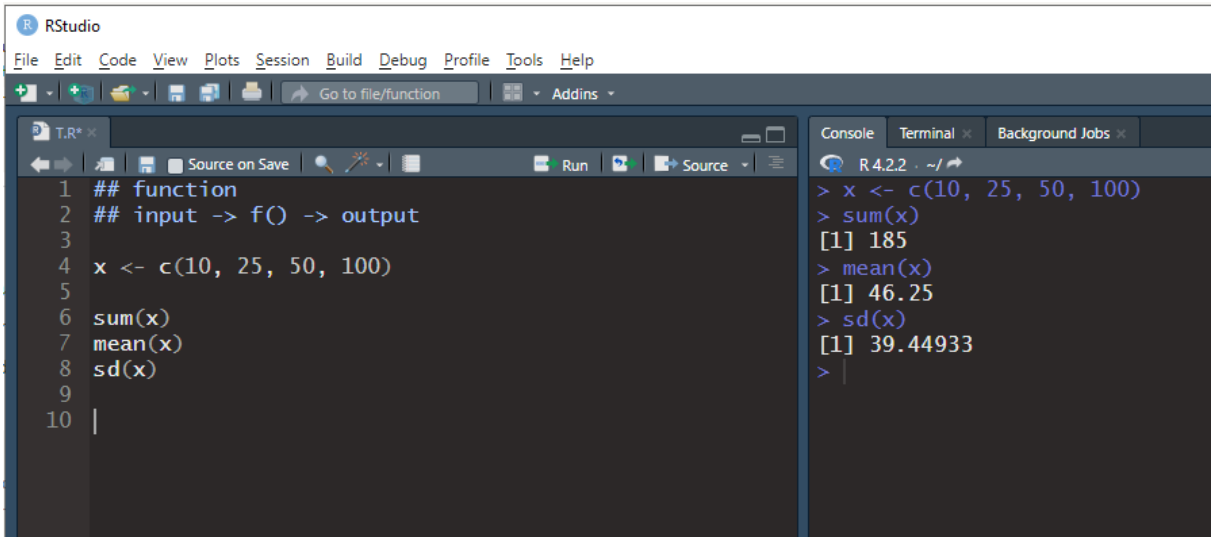
แก้ Code เพื่อให้วนแค่ 5 รอบ




```
count <- 0

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

Function

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

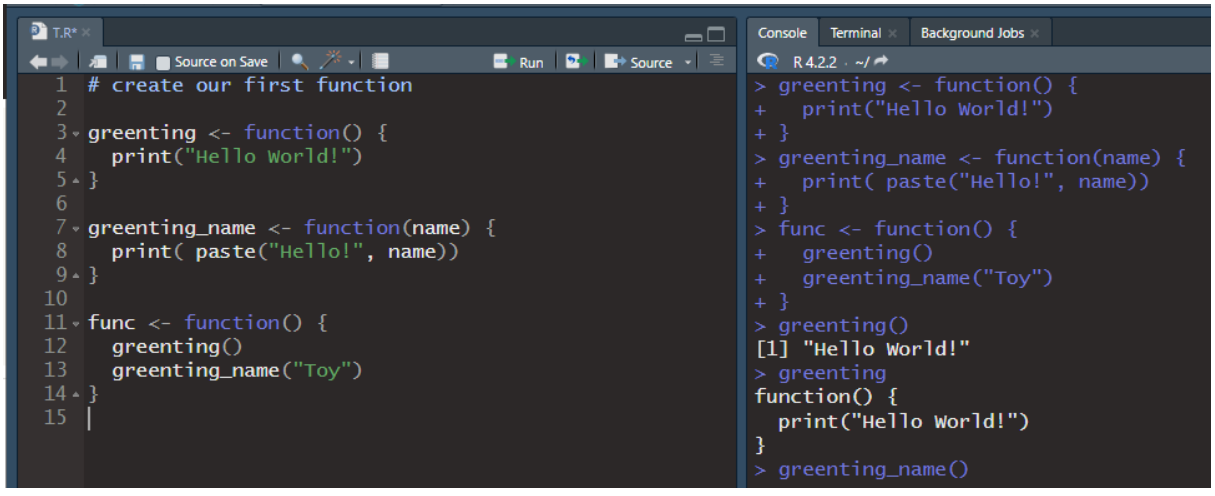


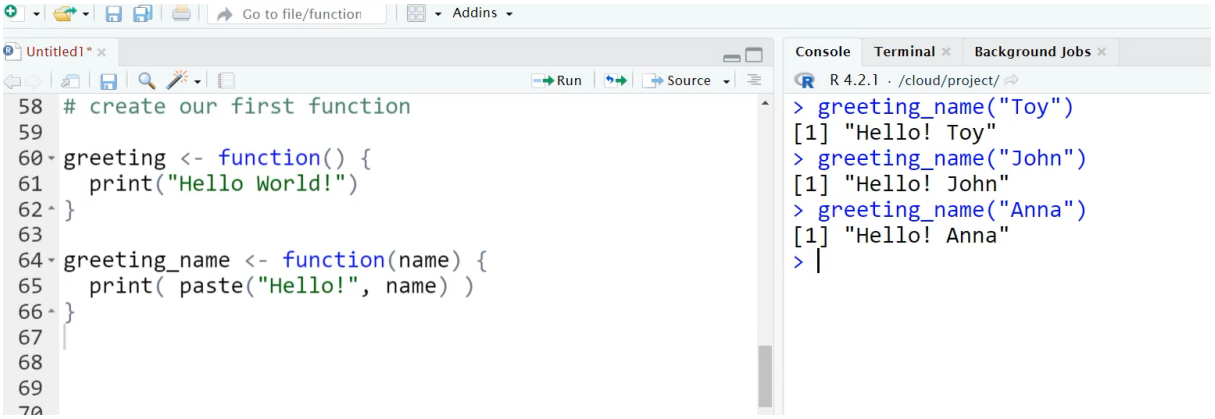

```
## function
## input -> f() -> output

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

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

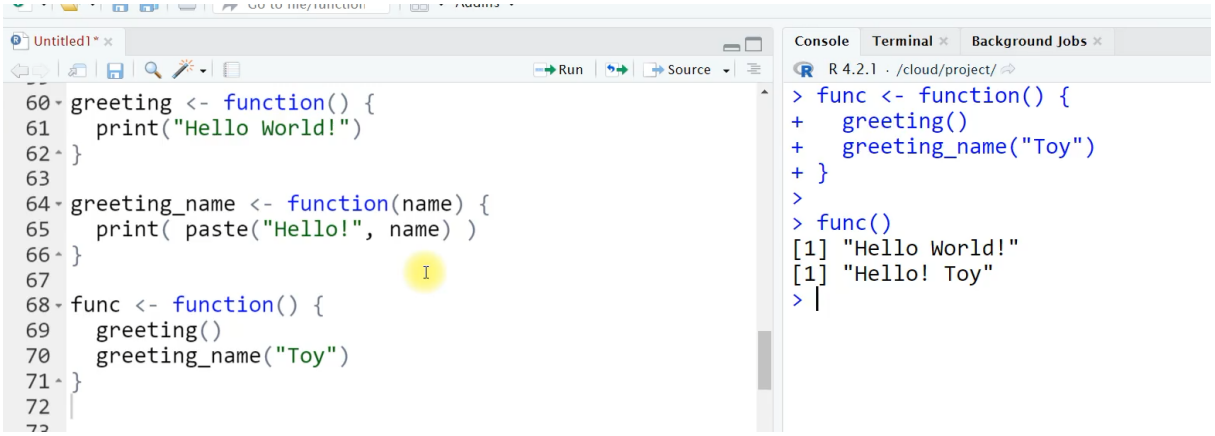
Create Our First Function

สร้าง Function ใช้เอง

R ใช้ keyword `function` ในการประกาศ function ใหม่








```
# create our first function

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

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

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

Function Parameter & Argument

คำศัพท์ที่เราใช้ในการเขียน `function` มีอยู่สองคำ (concept นี้ใช้ได้กับทุกภาษาเลย)

- `parameter`

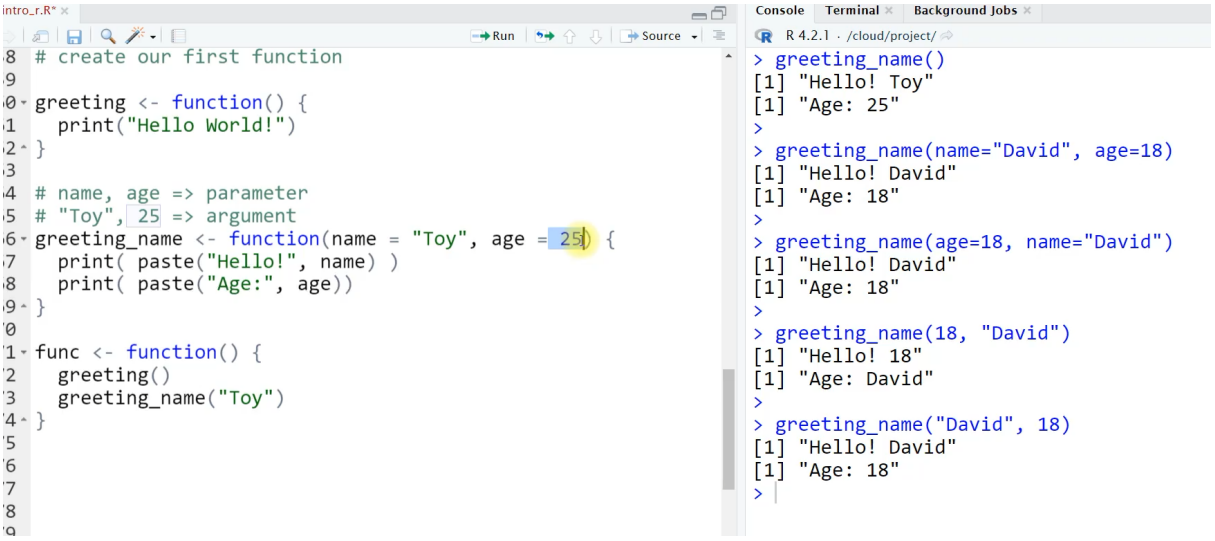
- `argument`

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 argumen




```
# 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

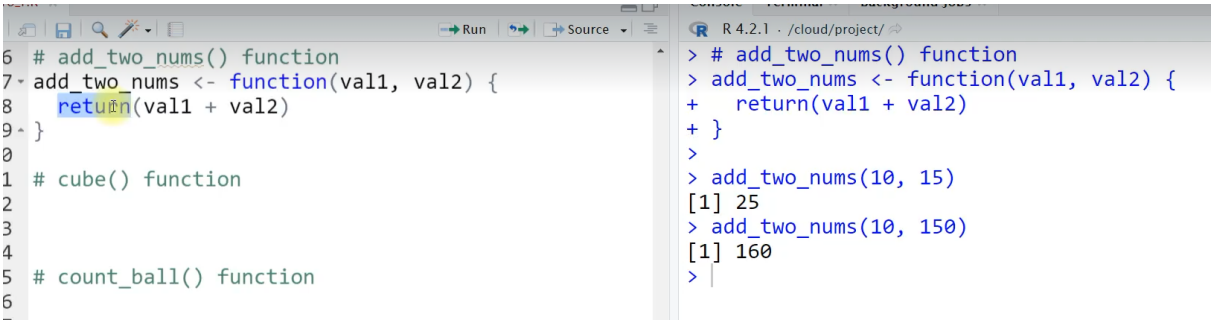
🌵 มาลองฝึกเขียน 3 functions ใน R ด้วยกันครับ

- `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



cube

เลขยกกำลัง

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

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

# count_ball() function
```

```
> 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

```r
# 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")
}
```

```
> 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 จำนวนของสีบอลแต่ละลูก

```r
# 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)
}
```

```
> 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 แสดงผลดีขึ้น หรือรันได้เร็วขึ้น แต่ผลลัพธ์ยังออกมาเหมือนเดิม

Function `cal_mean_by_col()` ที่แอดเขียนในวีดีโอแสดงผลไม่ค่อยสวยเท่าไหร่ แอดเลย `refactor` ใหม่ให้อ่านง่ายขึ้นด้วยโค้ดด้านล่าง

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

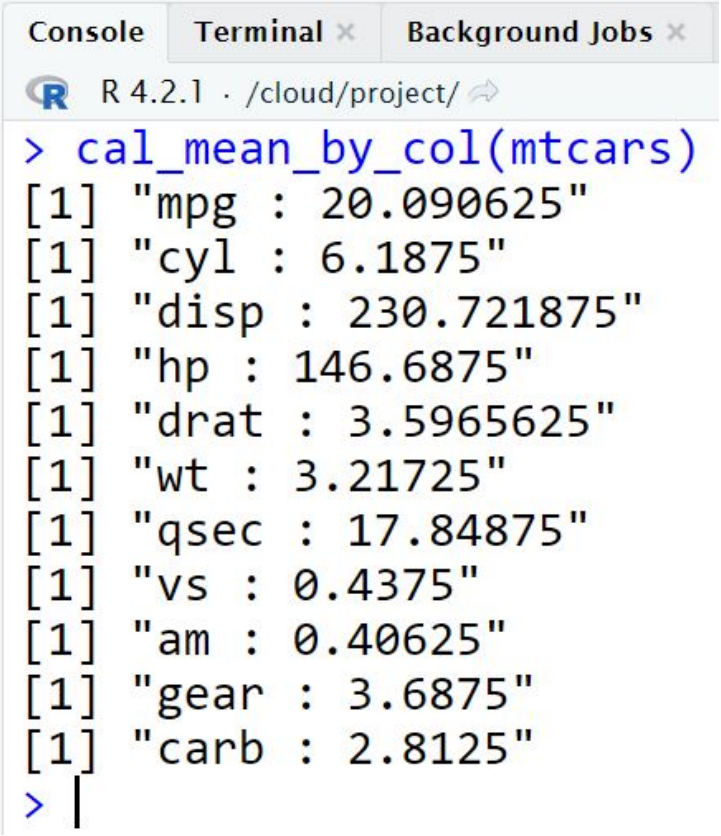
- `data$col_name`
- `data[["col_name"]]`
- `data[[5]]`

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)
```

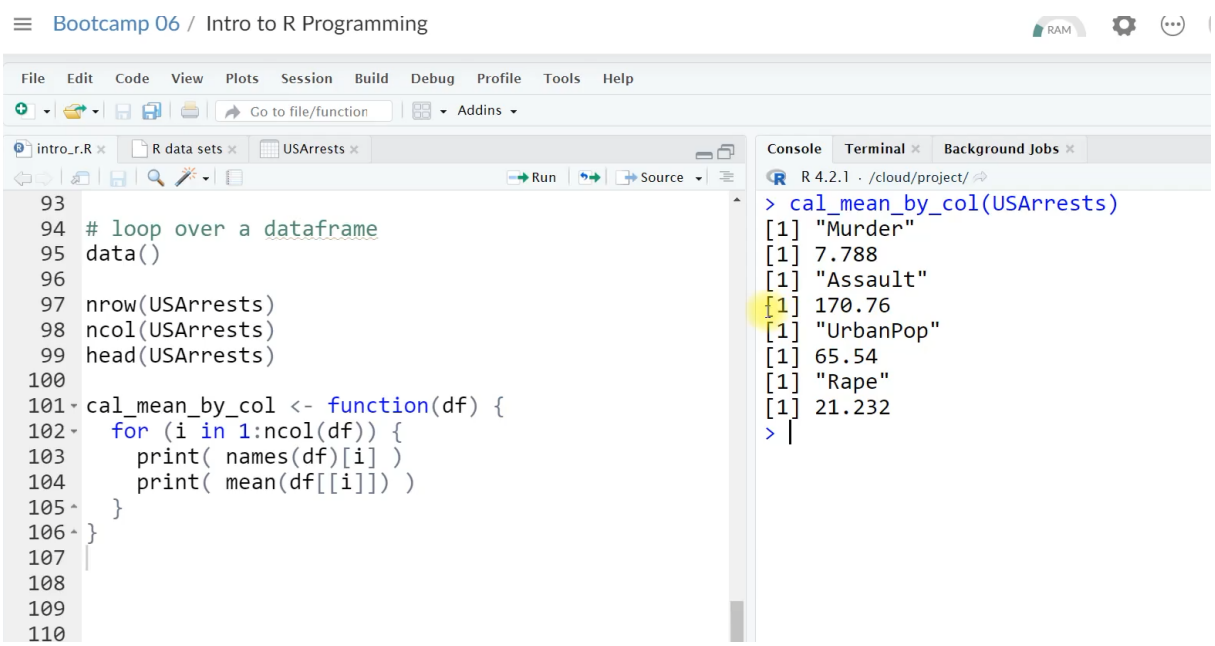
Output ที่ได้จาก `cal_mean_by_col()` ที่เรา refactor ใหม่ ดูง่ายขึ้นเยอะเลย 😝



ตัวอย่างข้อมูลจาก database

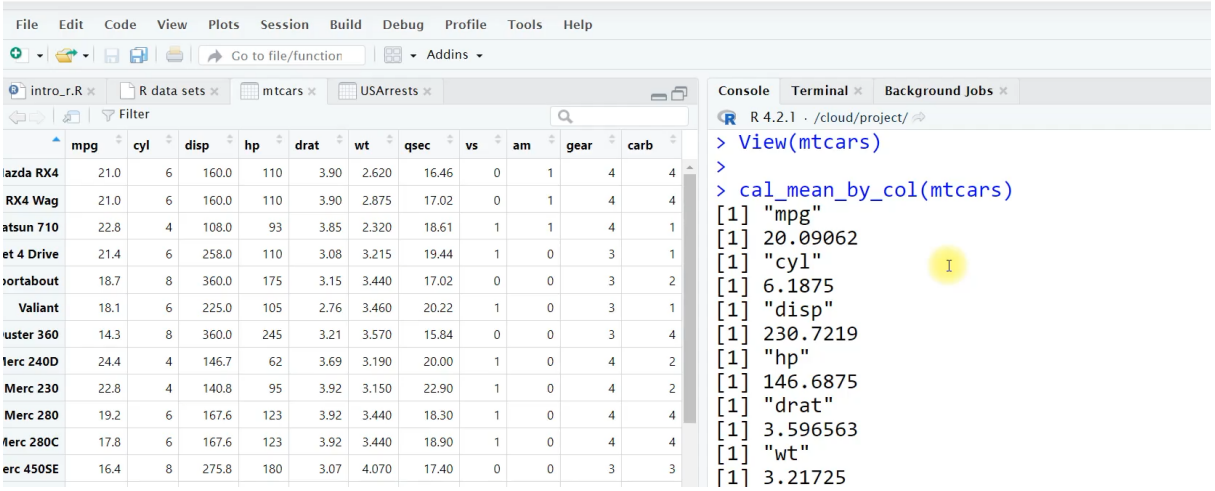
```
data()

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

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



ตัวอย่างที่ 2 Loop ข้อมูลที่เป็น mtcars

```
cal_mean_by_col(mtcars)
```

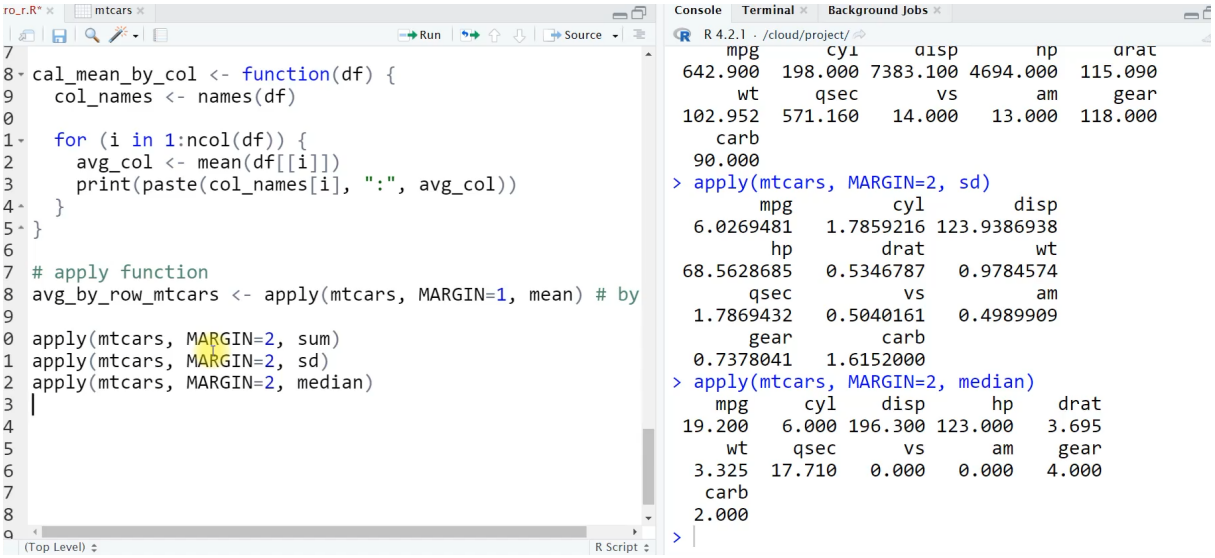


apply() coolest function

ใน R มีฟังก์ชัน `apply()` ที่เราใช้แทนการเขียน `loop` ได้ทั้ง columns, rows สะดวกมาก

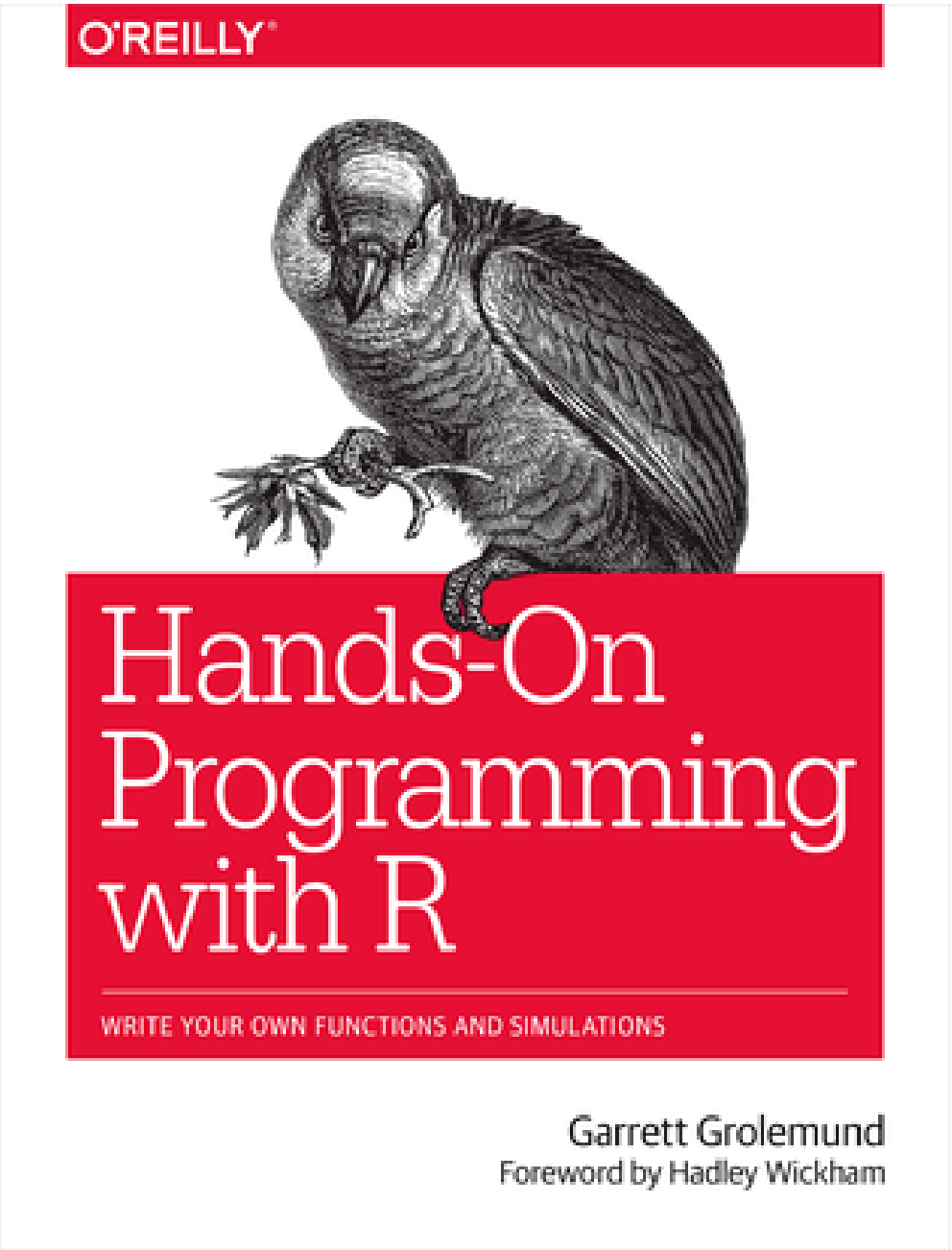
อ่านเพิ่มเติมเกี่ยว
กับ `apply()` ได้ที่ https://www.rdocumentation.org/packages/base/versions/3.6.2/topics/apply

```
# 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)
```



Reading - Hands On Programming with R

Reading Assignment



นักเรียนที่เรียนจบ `Control Flow and Function` แล้ว แอดมีการบ้านให้อ่านหนังสือ Hands-On Programming with R บทที่ 1-7

🌵 https://rstudio-education.github.io/hopr/

เสร็จแล้วเขียนสรุปใน `Notion` ของตัวเอง แชร์ได้ที่ห้อง #r ใน discord โรงเรียนเรานะครับ

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