

# R 데이터 분석 입문

6주차

## R 프로그래밍

오 세 종

 DANKOOK UNIVERSITY

# Contents

1. if 문
  2. 반복문 (for, while)
  3. 사용자정의 함수
  4. apply 계열 함수
  5. 프로그래밍 예제
- [R tip] 함수 내용보기, 파일에 출력하기

- R은 데이터 분석 도구인 동시에 프로그래밍 언어의 성격도 가지고 있다
- R 프로그래밍 기본 문법과 활용에 대해 학습 필요.
- 프로그래밍 : 주어진 문제(problem)를 컴퓨터가 해결(solution)하도록 하기 위한 절차(procedure)를 문법에 맞추어 서술하는 과정
- R 에서 제공하는 함수만으로는 문제를 해결할 수 없는 경우도 있는데, 이때 프로그래밍으로 문제를 해결한다

# 1. If 문

```
if (logical expression) {  
    statements  
} else {  
    alternative statements  
}
```

```
a <- 10  
if (a>5){  
    print (a)  
} else {  
    print (a*10)  
    print (a/10)  
}
```

```
> a = 10  
> if (a>5){  
+   print (a)  
+ } else {  
+   print (a*10)  
+   print (a/10)  
+ }  
[1] 10
```

else 는 앞의 } 와 같은 줄에 있어야함

# 1. If 문

```
a <- 10
b <- 20
if (a>5 & b>5) {                # and
  print (a+b)
}
if (a>5 | b>30) {               # or
  print (a*b)
}
```

```
> a <- 10
> b <- 20
> if (a>5 & b>5) {              # and
+   print (a+b)
+ }
[1] 30
> if (a>5 | b>30) {             # or
+   print (a*b)
+ }
[1] 200
```

# 1. Ifelse 문

```
a <- 10  
b <- 20  
ifelse (a>b, c<-a, c<-b)  
c
```

조건

조건이 참일때  
실행

조건이 거짓일때  
실행



```
a <- 10  
b <- 20  
if (a > b) {  
  c <- a  
} else {  
  c <- b  
}  
c
```

## 2. 반복문: for

```
for(i in 1:10) {  
  print(i)  
}
```

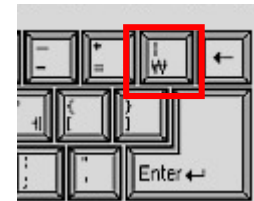
```
> for(i in 1:10) {  
+   print(i)  
+ }  
[1] 1  
[1] 2  
[1] 3  
[1] 4  
[1] 5  
[1] 6  
[1] 7  
[1] 8  
[1] 9  
[1] 10
```

## 2. 반복문: for

```
for(i in 1:10) {  
  cat("2*",i,"=",2*i,"\n")  
}
```

```
> for(i in 1:10) {  
+   cat("2*",i,"=",2*i,"\n")  
+ }  
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  
2* 10 = 20
```

\





## 2. 반복문: for

```
for(i in 1:20) {  
  if(i%%2==0) {  
    print(i)  
  }  
}
```

# 짝수인지 확인

```
> for(i in 1:20) {  
+   if(i%%2==0) {  
+     print(i)  
+   }  
+ }  
[1] 2  
[1] 4  
[1] 6  
[1] 8  
[1] 10  
[1] 12  
[1] 14  
[1] 16  
[1] 18  
[1] 20
```

## 2. 반복문: for

```
v1 <- 101:200
for(i in 1:length(v1)) {
  if(v1[i]%%2==0) {
    print(v1[i]*2)
  } else {
    print(v1[i]+2)
  }
}
```

```
[1] 103
[1] 204
[1] 105
[1] 208
[1] 107
[1] 212
[1] 109
[1] 216
[1] 111
[1] 220
[1] 113
[1] 224
[1] 115
```

## 2. 반복문: for

```
sum <- 0
for(i in 1:100) {
  sum <- sum + i
}
print(sum)
```

```
> sum <- 0
> for(i in 1:100) {
+   sum <- sum + i
+ }
> print(sum)
[1] 5050
```

## 2. 반복문: while

```
i<-1
while(i <= 10) {
  print(i)
  i <- i+1
}
```

```
> i<-1
>
> while(i <= 10) {
+   print(i)
+   i <- i+1
+ }
[1] 1
[1] 2
[1] 3
[1] 4
[1] 5
[1] 6
[1] 7
[1] 8
[1] 9
[1] 10
```

## 2. 반복문 : 예제

- iris 데이터셋에서 Sepal.Length 가 5.0~6.0 사이인 행들만 골라서 Sepal.Length, Sepal.Width 의 값을 보이시오

```
subset(iris, Sepal.Length >= 5.0 &  
        Sepal.Length <= 6.0) [,1:2]
```

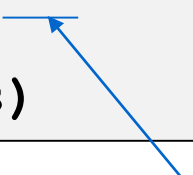
||

```
idx <- c()  
for (i in 1:nrow(iris)){  
  if (iris[i, "Sepal.Length"] >= 5.0 &  
      iris[i, "Sepal.Length"] <= 6.0) {  
    idx <- c(idx, i)  
  }  
}  
print(idx)  
iris[idx, c("Sepal.Length", "Sepal.Width")]
```

# Note

- 반복 횟수가 정해진 반복문은 for 를 이용하고, 조건에 따라 반복 횟수가 달라질 수 있는 반복문에는 while 을 사용한다
- 변수 초기화의 필요성

```
for(i in 1:100) {  
  ss <- ss + i      # error 발생  
}  
print(ss)
```



이 값이 무엇인지 알 수 없음

```
ss <- 0  
for(i in 1:100) {  
  ss <- ss + i      # 정상실행  
}  
print(ss)
```

## [연습문제 1]

1. 1~100 사이의 정수중 3의 배수들의 합과 개수를 구하시오
2. 101~200 사이의 숫자중 3과 4의 공배수를 출력하는 프로그램을 작성하시오
3. 24의 약수를 출력하는 프로그램을 작성하시오
4. 10! (팩토리얼) 을 출력하는 프로그램을 작성하시오
5. 0, 1, 1, 2, 3, 5, 8, 13, 21.....의 형태의 **수열**. 즉, 첫 번째 항의 값은 0이고 두 번째 항의 값은 1일 때 이후의 항들은 이전의 두 항을 더한 값으로 만들어지는 **수열**을 피보나치 수열이라고 한다. 0에서 부터 시작하여 40개의 피보나치 수열을 출력하시오

### 3. 사용자정의 함수 만들기

- 자주 반복적으로 하는 작업은 함수로 정의해 놓고 불러서 사용하는 것이 편리하다.
- R 은 다양한 함수를 패키지 형태로 제공하는데, 사용자도 함수를 정의하여 사용할 수 있다.
  - `sum()`, `max()`, `min()`, `barplot()`, `sd()`, ...



### 3. 사용자정의 함수 만들기

```
mymax <- function(x,y) {  
  num.max <- x  
  if (y > x) {  
    num.max <- y  
  }  
  return(num.max)  
}
```

```
mymax(10,15)  
mymax(20,15)
```

```
> mymax(10,15)  
[1] 15  
> mymax(20,15)  
[1] 20
```

### 3. 사용자정의 함수 만들기

함수의 이름

함수의 매개변수(parameter, argument)

```
mymax <- function(x,y) {  
  num.max <- x  
  if (y > x) {  
    num.max <- y  
  }  
  return(num.max)  
}
```

함수의 실행결과 값(return value)

### 3. 사용자정의 함수 만들기

- Default value

```
mydiv <- function(x, y=2) {  
  result <- x/y  
  return(result)  
}
```

default value 선언

```
mydiv(x=10, y=3)  
mydiv(10, 3)  
mydiv(10)
```

```
> mydiv(x=10, y=3)  
[1] 3.333333  
> mydiv(10, 3)  
[1] 3.333333  
> mydiv(10)  
[1] 5
```

### 3. 사용자정의 함수 만들기

- Return 해야 할 값이 하나가 아니고 여러 개 일 때

```
myfunc <- function(x,y) {  
  val.sum <- x+y  
  val.mul <- x*y  
  return(list(sum=val.sum, mul=val.mul))  
}
```

```
result <- myfunc(5,8)  
result$sum  
result$mul
```

```
> result$sum  
[1] 13  
> result$mul  
[1] 40
```

### 3. 사용자정의 함수 만들기

`sum {base}`

R Documentation

## Sum of Vector Elements

### Description

`sum` returns the sum of all the values present in its arguments.

### Usage

`sum(..., na.rm = FALSE)`

default value 선언

### Arguments

`...` numeric or complex or logical vectors.

`na.rm` logical. Should missing values (including `NaN`) be removed?

### 3. 사용자정의 함수 만들기

- 내가 정의한 함수가 저장된 파일 사용하기

```
mydiv <- function(x,y=2) {  
  result <- x/y  
  return(result)  
}
```

➡ myfunc.R

```
setwd("c:/Rworks")      # myfunc.R 이 저장된 폴더  
source("myfunc.R")      # myfunc.R 의 명령어 실행  
# 함수 사용  
a <- mydiv(20,4)  
b <- mydiv(30,4)  
a+b  
mydiv(mydiv(20,2),5)
```

## [연습문제 2]

1. 다음과 같이 두 정수를 입력하면 두 수의 최대 공약수를 찾아서 return 해 주는 함수 `lgm()` 를 작성하고 테스트 하시오

```
result <- lgm(10,8)
result
result <- lgm(10,20)
result
```

```
> result <- lgm(10,8)
> result
[1] 2
> result <- lgm(10,20)
> result
[1] 10
```

2. 다음과 같이 벡터를 입력하면 벡터의 최대값과 최소값을 return 하는 함수 `maxmin()` 을 작성하고 테스트 하시오 (return 값이 list 임)

```
v1 <- c(7,1,2,8,9)
result <- maxmin(v1)
result$max ; result$min
result <- maxmin(iris[,1])
result$max ; result$min
```

```
> v1 <- c(7,1,2,8,9)
> result <- maxmin(v1)
> result$max ; result$min
[1] 9
[1] 1
> result <- maxmin(iris[,1])
> result$max ; result$min
[1] 7.9
[1] 4.3
```

## 4. apply 함수

- R 프로그래밍에서는 for 나 while 을 사용하지 않는 것이 바람직하다. (처리속도 향상을 위하여)
- 대신 R 에서 제공하는 apply 계열 함수를 이용하면 다양한 반복문을 작성할 수 있다.
  - matrix, data frame, list 등에 있는 데이터에 대해 반복문을 적용해야 하는 경우는 대부분 apply 함수로 대체 가능



## 4. apply 함수

- apply()
  - matrix, data frame 에서 행단위, 열단위의 작업을 쉽게 할 수 있게 한다

```
for (i in 1:4) {  
  mean(iris[,i])  
}
```



```
apply(iris[,1:4], 2, mean)      # col 방향으로 함수적용
```

```
apply(iris[,1:4], 1, mean)     # row 방향으로 함수적용
```

\* 사용자 정의 함수도 적용할 수 있다

## 4. apply 함수

```
apply(iris[,1:4], 2, mean)
```

# col 방향으로 함수적용

Sepal.Length	Sepal.width	Petal.Length	Petal.width
5.1	3.5	1.4	0.2
4.9	3.0	1.4	0.2
4.7	3.2	1.3	0.2
4.6	3.1	1.5	0.2
5.0	3.6	1.4	0.2
5.4	3.9	1.7	0.4
4.6	3.4	1.4	0.3
5.0	3.4	1.5	0.2
4.4	2.9	1.4	0.2
4.9	3.1	1.5	0.1
5.4	3.7	1.5	0.2
4.8	3.4	1.6	0.2
4.8	3.0	1.4	0.1
4.3	3.0	1.1	0.1
5.8	4.0	1.2	0.2

```
apply(iris[,1:4], 1, mean)
```

# row 방향으로 함수적용

Sepal.Length	Sepal.width	Petal.Length	Petal.width
5.1	3.5	1.4	0.2
4.9	3.0	1.4	0.2
4.7	3.2	1.3	0.2
• 4.6	3.1	1.5	0.2
5.0	3.6	1.4	0.2
5.4	3.9	1.7	0.4
• 4.6	3.4	1.4	0.3
5.0	3.4	1.5	0.2
4.4	2.9	1.4	0.2
• 4.9	3.1	1.5	0.1
5.4	3.7	1.5	0.2
4.8	3.4	1.6	0.2
• 4.8	3.0	1.4	0.1
4.3	3.0	1.1	0.1
5.8	4.0	1.2	0.2

## 4. apply 함수

```
> apply(iris[,1:4], 2, mean)
Sepal.Length Sepal.Width Petal.Length Petal.Width
5.843333      3.057333      3.758000      1.199333
> apply(iris[,1:4], 1, mean)
 [1] 2.550 2.375 2.350 2.350 2.550 2.850 2.425 2.525 2.225 2.400 2.700 2.500 2.325 2.125 2.800
[16] 3.000 2.750 2.575 2.875 2.675 2.675 2.675 2.350 2.650 2.575 2.450 2.600 2.600 2.550 2.425
[31] 2.425 2.675 2.725 2.825 2.425 2.400 2.625 2.500 2.225 2.550 2.525 2.100 2.275 2.675 2.800
[46] 2.375 2.675 2.350 2.675 2.475 4.075 3.900 4.100 3.275 3.850 3.575 3.975 2.900 3.850 3.300
[61] 2.875 3.650 3.300 3.775 3.350 3.900 3.650 3.400 3.600 3.275 3.925 3.550 3.800 3.700 3.725
[76] 3.850 3.950 4.100 3.725 3.200 3.200 3.150 3.400 3.850 3.600 3.875 4.000 3.575 3.500 3.325
[91] 3.425 3.775 3.400 2.900 3.450 3.525 3.525 3.675 2.925 3.475 4.525 3.875 4.525 4.150 4.375
[106] 4.825 3.400 4.575 4.200 4.850 4.200 4.075 4.350 3.800 4.025 4.300 4.200 5.100 4.875 3.675
[121] 4.525 3.825 4.800 3.925 4.450 4.550 3.900 3.950 4.225 4.400 4.550 5.025 4.250 3.925 3.925
[136] 4.775 4.425 4.200 3.900 4.375 4.450 4.350 3.875 4.550 4.550 4.300 3.925 4.175 4.325 3.950
>
```

## 4. apply 함수

- lapply()
  - apply() 와 유사하나 결과가 list format 이다

```
lapply(iris[,1:4], mean)      # col 방향으로 함수적용됨
```

```
> apply(iris[,1:4], 2, mean)
Sepal.Length Sepal.Width Petal.Length  Petal.Width
      5.843333      3.057333      3.758000      1.199333
```

➡ 결과가 vector

```
>
> lapply(iris[,1:4], mean)
$Sepal.Length
[1] 5.843333
```

```
$Sepal.Width
[1] 3.057333
```

```
$Petal.Length
[1] 3.758
```

```
$Petal.Width
[1] 1.199333
```

➡ 결과가 list

## 4. apply 함수

- `apply()`
  - 실행 결과를 vector 또는 list 로 선택할 수 있다

```
apply(iris[,1:4], mean) # 결과가 vector
```

```
apply(iris[,1:4], mean, simplify=F) # 결과가 list
```

```
> apply(iris[,1:4], mean) # 결과가 vector
Sepal.Length Sepal.Width Petal.Length Petal.Width
      5.843333      3.057333      3.758000      1.199333
>
> apply(iris[,1:4], mean, simplify=F) # 결과가 list
$Sepal.Length
[1] 5.843333

$Sepal.Width
[1] 3.057333

$Petal.Length
[1] 3.758

$Petal.Width
[1] 1.199333
```

## 4. apply 함수

- 사용자 정의 함수를 적용한 sapply

```
myfunc <- function (x) {  
  a <- max(x)  
  b <- min(x)  
  return(a-b)  
}  
  
sapply(iris[,1:4], myfunc)
```

```
> myfunc <- function (x) {  
+   a <- max(x)  
+   b <- min(x)  
+   return(a-b)  
+ }  
>  
> sapply(iris[,1:4], myfunc)  
Sepal.Length Sepal.Width Petal.Length Petal.Width  
          3.6         2.4         5.9         2.4
```

## 5. 프로그래밍 예제

- 화면에서 사용자 입력값 받기

```
n <- readline(prompt="숫자를 입력하세요: ")
cat("입력한 숫자는", n, "입니다. \n")
```

```
> n <- readline(prompt="숫자를 입력하세요: ")
숫자를 입력하세요: 7
> cat("입력한 숫자는", n, "입니다. \n")
입력한 숫자는 7 입니다.
```

## 5. 프로그래밍 예제

- 숫자 맞추기 게임

```
num <- round(runif(1) * 100, digits = 0)
guess <- -1
cat("Guess a number between 0 and 100.\n")

while(guess != num){
  guess <- readline(prompt="Guess number :")
  guess <- as.integer(guess)
  if (guess == num) {
    cat("Congratulations,", num, "is right.\n")
  } else if (guess < num){
    cat("It's smaller!\n")
  } else if(guess > num) {
    cat("It's bigger!\n")
  }
}
```



## [연습 3]

1. 세개의 숫자를 매개변수로 입력하면 그중에 가장 큰 수를 돌려주는 함수를 작성하고 테스트 하시오
2. 화면에서 숫자 2개를 입력 받아 두숫자의 합과 곱을 출력하는 프로그램을 작성하시오 (이작업을 계속 반복하되 두 숫자가 모두 0 이면 프로그램을 중지한다)

## [R Tip] 함수의 내용 보기

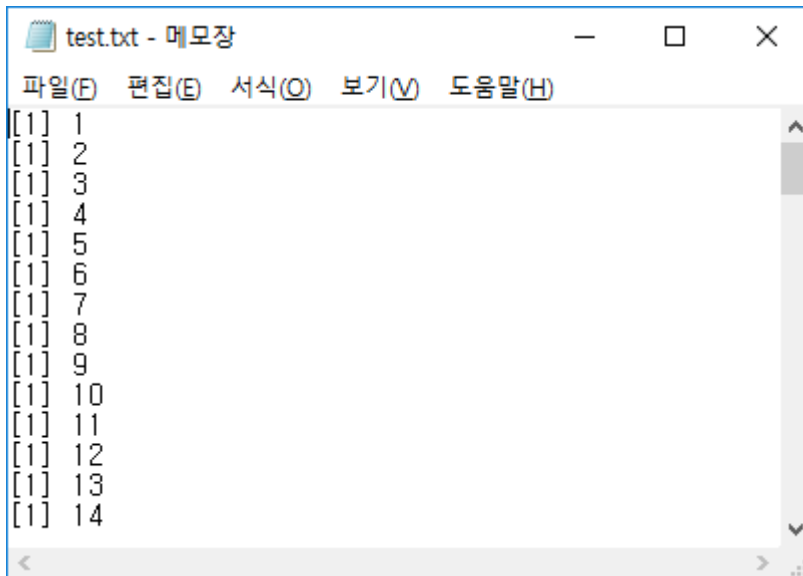
- R에서 제공하는 함수의 source code 보기 : 함수의 이름을 () 없이 입력

```
> sd
function (x, na.rm = FALSE)
sqrt(var(if (is.vector(x) || is.factor(x)) x else as.double(x),
  na.rm = na.rm))
<bytecode: 0x000000001d0ab468>
<environment: namespace:stats>
> apply
function (X, MARGIN, FUN, ...)
{
  FUN <- match.fun(FUN)
  d1 <- length(dim(X))
  if (!d1)
    stop("dim(X) must have a positive length")
  if (is.object(X))
    X <- if (d1 == 2L)
      as.matrix(X)
    else as.array(X)
  d <- dim(X)
  dn <- dimnames(X)
  ds <- seq_len(d1)
  if (is.character(MARGIN)) {
    if (is.null(dnn <- names(dn)))
      stop("'X' must have named dimensions")
  }
}
```

# [R Tip] 실행 내용 파일로 출력하기

- text.txt 파일에 1~100 숫자를 출력해 보자

```
sink("test.txt", append=T) # 출력할 파일 open
for (i in 1:100) {
    print(i)                  # 지정한 파일로 출력
}
sink()                     # 출력할 파일 close
```



append=T : 지정한 파일에  
이미 어떤 내용이 있으면  
내용 뒤에 이어서 저장.  
append=F : 지정한 파일에  
이미 어떤 내용이 있으면  
내용을 삭제하고 새로 저장.

실습x