

# R-프로그래밍

서울대학교 통계연구소

2023년 8월

# 이번 강의에서 다룰 내용

- 자료저장 및 외부 데이터 불러오기
- 함수 작성에 필요한 제어문, 반복문
- 함수 작성 및 응용
- 효율적인 프로그래밍 방안

## R Working directory

- R에서 생성한 데이터나 분석한 결과를 컴퓨터에 저장하거나 컴퓨터에 있는 데이터 파일을 R에 불러올 수 있다.
- 먼저 컴퓨터의 어디에(directory) 저장을 할 지, 어디에서 불러 올지를 생각해야 한다.
- R이 실행되었을때의 작업 디렉토리(working directory)를 확인하는 방법으로 `getwd()`를 사용하고, 작업디렉토리를 원하는 디렉토리로 바꾸는 작업은 `setwd()`를 이용한다.

## scan()을 이용한 자료 읽기

- 다음과 같은 내용이 각각 들어 있는 txt파일들을 만들어보자

- z1.txt

123

4 5

6

- z2.txt

123

4.2 5

6

- z3.txt

abc

de f

g

- z4.txt

abc

123 6

y

각 파일을 다음과 같이 읽어보자.

```
scan("z1.txt")
```

```
## [1] 123 4 5 6
```

```
scan("z2.txt")
```

```
## [1] 123.0 4.2 5.0 6.0
```

```
scan("z3.txt")
```

```
## Error in scan("z3.txt"): scan() expected 'a real', got 'abc'
```

```
scan("z3.txt", what = "")
```

```
## [1] "abc" "de" "f" "g"
```

```
scan("z4.txt", what = "")
```

```
## [1] "abc" "123" "6"  "y"
```

```
scan("z3.txt", what = "")
```

```
## [1] "abc" "de"  "f"   "g"
```

```
scan("z3.txt", what = "", sep = "\n")
```

```
## [1] "abc"  "de f" "g"
```

# 스크린에 결과값 보여주기

```
x <- 1:3  
print(x^2)
```

```
## [1] 1 4 9
```

```
print("abc")
```

```
## [1] "abc"
```



```
cat("abc\n")
```

```
## abc
```

```
cat(x, "abc", "de\n")
```

```
## 1 2 3 abc de
```

```
cat(x, "abc", "de\n", sep = "")
```

```
## 123abcde
```

## R 결과물 저장 및 읽기

- CSV: comma separated values
- 많은 데이터 파일이 CSV 형식으로 되어 있고, R에서도 이 형식의 파일 저장 및 읽기가 지원된다.
- 다음과 같은 자료를 CSV 파일로 저장해보자.

id	name	score
1	Mr. Foo	95
2	Ms. Bar	97
3	Mr. Baz	92

## 저장

```
id = c(1, 2, 3)
name = c('Mr. Foo', 'Ms. Bar', 'Mr. Baz')
score = c(95, 97, 92)
a = data.frame(id, name, score)

write.csv(a, file = 'a.csv')
write.csv(a, file = 'a2.csv', row.names = FALSE)
write.table(a, quote = FALSE, sep = ',',
            file = 'a3.csv', row.names = FALSE)
write.table(a, quote = FALSE, sep = '\t',
            file = 'a4.txt', row.names = FALSE)
```

작업디렉토리(working directory)에 파일들이 생성된것을 알수 있다.

# 읽기

```
x <- read.csv("a2.csv")
```

```
x
```

```
##      id      name score
## 1    1 Mr. Foo    95
## 2    2 Ms. Bar    97
## 3    3 Mr. Baz    92
```

```
str(x)
```

```
## 'data.frame':    3 obs. of  3 variables:
##  $ id      : int  1 2 3
##  $ name    : chr  "Mr. Foo" "Ms. Bar" "Mr. Baz"
##  $ score   : int  95 97 92
```

첫 줄(header)을 없애고 파일로 저장하고 읽어보자.

1	Mr. Foo	95
2	Ms. Bar	97
3	Mr. Baz	92

```
write.table(a, quote = FALSE, sep = ',', file = 'b.csv',  
            row.names = FALSE, col.names = FALSE)
```

```
y <- read.csv("b.csv", header = FALSE)  
y
```

```
##      V1      V2 V3  
## 1  1 Mr. Foo 95  
## 2  2 Ms. Bar 97  
## 3  3 Mr. Baz 92
```

읽어들인 자료의 변수들에 이름을 지정해보자.

```
colnames(y)
```

```
## [1] "V1" "V2" "V3"
```

```
colnames(y) <- c('id', 'name', 'score')
```

```
y
```

```
##   id   name score
## 1  1 Mr. Foo    95
## 2  2 Ms. Bar    97
## 3  3 Mr. Baz    92
```

문자열 (chr) 대신 요인 (factor) 변수형으로 파일을 읽으려면  
stringsAsFactors = TRUE 옵션을 준다.

```
z <- read.csv("a2.csv", stringsAsFactors = TRUE)
str(z)
```

```
## 'data.frame':    3 obs. of  3 variables:
## $ id      : int  1 2 3
## $ name    : Factor w/ 3 levels "Mr. Baz","Mr. Foo",...: 2 3 1
## $ score   : int  95 97 92
```

# R object 저장하기

- 여러개의 R object 를 하나의 파일로 저장해보자.

```
b = list(a = 1:3, b = TRUE, c = 'oops')  
save(a, b, file = 'xy.RData')  
load('xy.RData')
```

- 작업디렉토리(working directory)에 “xy.RData” 파일이 만들어 진 것을 볼 수 있다. `load("xy.RData")` 명령어를 통해 저장된 데이터를 불러올 수 있다.
- 파일에 저장되어 있는 R object가 R에 불러졌는지 확인하기 위해 `ls()`를 사용해보자.
- 필요없는 R object는 `rm()`으로 지울수 있다.



# 인터넷상 자료를 읽어들이기



웹에 있는 파일의 경우 URL을 지정하여 바로 읽어올 수 있다.













## UCI Machine Learning Repository



### Welcome to the UC Irvine Machine Learning Repository!

We currently maintain 488 data sets as a service to the machine learning community. You may [view all data sets](#) through our searchable interface. For a general overview of the Repository, please visit our [About](#) page. For information about citing data sets in publications, please read our [citation policy](#). If you wish to donate a data set, please consult our [donation policy](#). For any other questions, feel free to [contact the Repository librarians](#).

Supported By:  In Collaboration With: 

Latest News:	Newest Data Sets:	Most Popular Data Sets (hits since 2007):
<p><b>09-24-2018:</b> Welcome to the new Repository admins Dheeru Dua and Eli Kira Taniskidou!</p> <p><b>04-04-2013:</b> Welcome to the new Repository admins Kevin Bache and Moshe Lichman!</p> <p><b>03-01-2010:</b> Note from donor regarding Netflix data</p> <p><b>10-16-2009:</b> Two new data sets have been added.</p> <p><b>09-14-2009:</b> Several data sets have been added.</p> <p><b>03-24-2008:</b> New data sets have been added!</p> <p><b>06-25-2007:</b> Two new data sets have been added: UJI Pen Characters, MAGIC Gamma Telescope</p>	<p><b>10-06-2019:</b>  <a href="#">WISDM Smartphone and Smartwatch Activity and Biometrics Dataset</a></p> <p><b>09-30-2019:</b>  <a href="#">Hepatitis C Virus (HCV) for Egyptian patients</a></p> <p><b>09-23-2019:</b>  <a href="#">QSAR fish toxicity</a></p> <p><b>09-23-2019:</b>  <a href="#">QSAR aquatic toxicity</a></p> <p><b>09-21-2019:</b>  <a href="#">Online Retail II</a></p> <p><b>09-20-2019:</b>  <a href="#">Human Activity Recognition from Continuous Ambient Sensor Data</a></p>	<p><b>3062854:</b>  <a href="#">Iris</a></p> <p><b>1695033:</b>  <a href="#">Adult</a></p> <p><b>1314025:</b>  <a href="#">Wine</a></p> <p><b>1112775:</b>  <a href="#">Wine Quality</a></p> <p><b>1111362:</b>  <a href="#">Heart Disease</a></p> <p><b>1106612:</b>  <a href="#">Car Evaluation</a></p>

UCI Machine Learning Repository 에서 레드와인 등급자료 불러오기  
<https://archive.ics.uci.edu/ml/machine-learning-databases/wine-quality/winequality-red.csv>

```
addr <- "https://archive.ics.uci.edu/ml/machine-learning-databases/wine-quality/winequality-red.csv"
w = read.csv(addr, header = TRUE, sep = ";")
w[1:6, 1:3]
```

##	fixed.acidity	volatile.acidity	citric.acid
## 1	7.4	0.70	0.00
## 2	7.8	0.88	0.00
## 3	7.8	0.76	0.04
## 4	11.2	0.28	0.56
## 5	7.4	0.70	0.00
## 6	7.4	0.66	0.00

## 반복문 Loops (for, while, if)

- 프로그래밍을 간단하게 만들어 준다.
- for문과 while문
- break: 반복문의 진행을 즉시 종료.
- next: 현재 단계의 반복을 생략하고 다음 인덱스로 넘어감.
- break와 next는 예제를 보면 쉽게 이해할 수 있음.

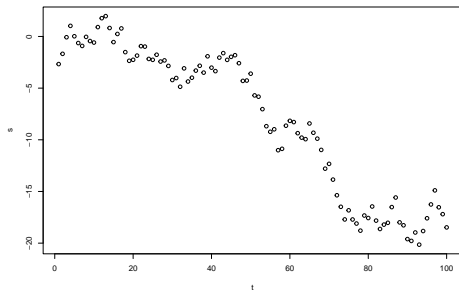
```
for(var in seq) { expr }
```

- 지정된 변수(var)가 주어진 벡터열(seq)의 값 하나하나에 대해 문장(expr)을 반복 실행한다.
- e.g., for 을 이용하여 부분합 만들기 (다음 페이지)

$$s_t = \sum_{i=1}^t e_i \text{ for } t = 1, \dots, n \text{ where } e_i \sim \mathcal{N}(0, 1)$$

for

```
e = rnorm(100)
n = length(e)
s = rep(0, n)
for (i in 1:n) s[i] = sum(e[1:i])
t = 1:n
plot(t, s)
```



`while(cond) expr`

- 주어진 조건문(cond)이 참인 경우에 문장(expr)을 반복 실행
- e.g., 수열의 값이 처음 음수인 위치 찾기

```
b = c(1, 5, 8, 0, -1, 2)
counter = 0
isPositive = TRUE
while (isPositive) {
  counter = counter + 1
  isPositive = (b[counter] >= 0)
}
cat("A negative number is detected at the",
    counter, "th place.\n")
```

```
## A negative number is detected at the 5 th place.
```

# break

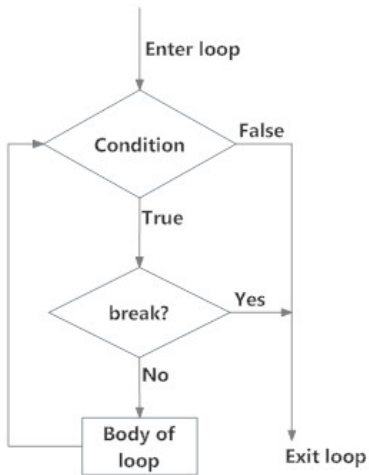


Fig: flowchart of break

- 1부터 5까지의 수가 뒤섞인 수열  $x$  에서 값이 4인 곳의 위치 찾기

```
(x <- sample(1:5, 5))
```

```
## [1] 1 4 3 5 2
```

```
counter = 0
for (val in x) {
  counter = counter + 1
  if (val == 4){
    cat("x is 4 at the", counter, "th place.\n")
    break
  }
}
```

```
## x is 4 at the 2 th place.
```



next

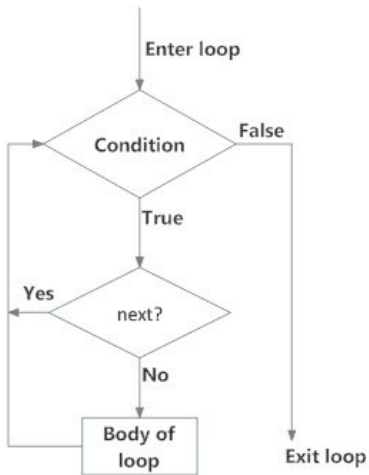


Fig: flowchart of next

- 수열  $x$  에서 3보다 작거나 같은 값의 위치 찾기

```
(x <- sample(1:5, 5))
```

```
## [1] 1 4 5 3 2
```

```
counter = 0
for (val in x) {
  counter = counter + 1
  if (val <= 3){
    cat("x is less than 3 at the", counter, "th place.\n")
    next
  }
}
```

```
## x is less than 3 at the 1 th place.
```

```
## x is less than 3 at the 4 th place.
```

```
## x is less than 3 at the 5 th place.
```

## if(cond) expr1 else expr2

- 주어진 조건 (cond) 이 참인 경우 expr1 을, 거짓인 경우에는 else 뒤의 expr2 을 실행한다.
- e.g.,  $x = 3$  이면  $y = 1$ , 아니면  $y = -1$

```
x = 3
if (x > 1) {
  y = 1
} else {
  y = -1
}
y
```

```
## [1] 1
```

```
# c.f., ifelse (test, yes, no)
(y = ifelse(x > 1, 1, -1))
```

```
## [1] 1
```

# R Functions

`function( arglist ) expr`

- R에서 함수란 복잡한 계산식 (expr) 을 매번 코딩할 필요 없이 입력변수 (arglist) 만 바꾸어가며 반복적으로 사용할 수 있는 코드 뭉치를 말한다.
- 내장 함수: 기본적으로 주어지는 R 함수들

e.g.,  $f(\mathbf{x}) = \frac{1}{n} \sum_{i=1}^n x_i$

```
myvector = c(1, 1, 1, 2, 2, 2, 2)
mean(myvector)
```

```
## [1] 1.571429
```

- 사용자 정의 함수

e.g.,  $f(x) = 20 + x^2$

```
myfunction <- function(x) { 20 + x * x }  
myfunction(10)
```

```
## [1] 120
```

```
myfunction(25)
```

```
## [1] 645
```

- 미정 함수

```
fish(myvector)
```

```
## Error in fish(myvector): could not find function "fish"
```

- fish() 함수가 정의되지 않았으므로 에러메세지를 보냄.
- R에 존재하지 않는 명령을 실행해도 컴퓨터가 고장나지 않고 조용히 에러를 출력함.
- 에러 메세지를 살펴보면 많은 경우 문제를 파악하고 해결할 수 있음.

# R 함수 만들기 1

- R 은 기본적으로 평균 함수 `mean()`, 중앙값 함수 `median()` 등은 내장하고 있지만, 최빈값(mode)을 계산하는 함수는 갖고 있지 않다.
- R 내장함수 `mode()`는 다른 일을 한다.
- 그러므로 최빈값을 계산하는 함수를 직접 만들어 보자.
- 물론, 첫번째 수업에서 다룬 `modeest`라는 패키지를 이용할 수도 있다.

- 최빈값 구하기

```
x1 <- c(1, 2, 1, 2, 2, 3, 3, 4, 5, 4, 5)
(count_x1 <- tabulate(x1))
```

```
## [1] 2 3 2 2 2
```

```
which.max(count_x1)
```

```
## [1] 2
```

- 새로운 수열이 주어지면 같은 작업을 반복해야한다.

```
x2 <- c(3, 4, 7, 2, 3, 3, 7, 3, 2, 6, 6)
count_x2 <- tabulate(x2)
which.max(count_x2)
```

```
## [1] 3
```



- 최빈값 함수 만들기

```
MyMode <- function(xval){  
  count_xval <- tabulate(xval)  
  res <- which.max(count_xval)  
  return(res)  
}
```

- 입력값만 바꾸어주면 최빈값이 계산된다.

```
MyMode(x1) # x1 <- c(1, 2, 1, 2, 2, 3, 3, 4, 5, 4, 5)
```

```
## [1] 2
```

```
MyMode(x2) # x2 <- c(3, 4, 7, 2, 3, 3, 7, 3, 2, 6, 6)
```

```
## [1] 3
```

## R 함수 만들기 2

- 이번엔 입력변수가 여러개이고 옵션에 따라 output이 달라지도록 함수를 만들어보자.
- R에서 `which.max()`는 입력벡터의 값들중에 최댓값이 위치해 있는 index를 리턴해주는 함수이다.

```
mdata = c(1, 2, 1, 3, 4, 9, 5)
which.max(mdata)
```

```
## [1] 6
```

```
mdata = c(1, 2, 1, 3, 4, 9, 5, 9)
which.max(mdata)
```

```
## [1] 6
```

- 최댓값이 여러군데에 있는 경우, 최초의 index만 리턴해준다.
- 옵션에 따라 최댓값을 갖는 모든 index를 리턴해주는 함수를 만들어보자. (`all`)

```

mywhich.max = function(x, val = FALSE, all = FALSE) {
  n = length(x); ind = 1; m = x[1]
  for (i in 2:n) { if (m < x[i]) { ind = i; m = x[i] } }
  all.ind = (1:n)[x == m]

  if (val == TRUE) {
    if (all == TRUE) {
      return(list(max.ind = all.ind, max.val = m))
    } else {
      return(list(max.ind = ind, max.val = m))
    }
  } else {
    if (all == TRUE) {
      return(list(max.ind = all.ind))
    } else {
      return(list(max.ind = ind))
    }
  }
}

```

```
mywhich.max(mdata)
```

```
## $max.ind  
## [1] 6
```

```
mywhich.max(mdata, all = TRUE)
```

```
## $max.ind  
## [1] 6 8
```

```
mywhich.max(mdata, val = TRUE)
```

```
## $max.ind  
## [1] 6  
##  
## $max.val  
## [1] 9
```

```
mywhich.max(mdata, val = TRUE, all = TRUE)
```

```
## $max.ind  
## [1] 6 8  
##  
## $max.val  
## [1] 9
```

mywhich.max는 벡터를 입력변수로 받아들인다. 만약 다른 R object를 넣으면 어떻게 될까?

```
(mdata2 = rbind(mdata, mdata))
```

```
##           [,1] [,2] [,3] [,4] [,5] [,6] [,7] [,8]
## mdata      1    2    1    3    4    9    5    9
## mdata      1    2    1    3    4    9    5    9
```

```
mywhich.max(mdata2)
```

```
## $max.ind
## [1] 11
```

```
mywhich.max(mdata2, all = TRUE)
```

```
## $max.ind
## [1] 11 12 15 16
```

## 벡터가 아닌 입력변수의 경우 에러메세지를 내보자

```
mywhich.max = function(x, val = FALSE, all = FALSE) {  
  if (!is.vector(x)) stop('input is not a vector!')  
  n = length(x); ind = 1; m = x[1]  
  for (i in 2:n) { if (m < x[i]) { m = x[i]; ind = i } }  
  all.ind = (1:n)[x == m]  
  if (val == TRUE) {  
    if (all == TRUE) {  
      return(list(max.ind = all.ind, max.val = m))  
    } else {  
      return(list(max.ind = ind, max.val = m))  
    }  
  } else {  
    if (all == TRUE) {  
      return(list(max.ind = all.ind))  
    } else {  
      return(list(max.ind = ind))  
    } } }
```

```
mywhich.max(mdata2)
```

```
## Error in mywhich.max(mdata2): input is not a vector!
```

stop() 대신 warning()를 사용할 수도 있다.



# R 함수의 입력

임의의 함수도 다른 함수의 input으로 사용할 수 있다.

```
f1 = function(a, b)
  return(a + b)
f2 = function(a, b)
  return(a - b)
g = function(h, a, b)
  h(a, b) # h 는 임의의 함수
g(f1, 3, 2)
```

```
## [1] 5
```

```
g(f2, 3, 2)
```

```
## [1] 1
```

## R 함수의 결과

함수의 output이 특정 함수가 되게 할 수 있다.

```
g = function(y) {  
  h = function(x) {  
    return(x ^ 2 + y)  
  }  
  return(h)  
}
```

```
test.ft = g(1) # test.ft가 g()의 output인 h 함수 역할을 함  
test.ft(2) # 2^2 + 1
```

```
## [1] 5
```

## 함수를 여러개 만들 때

- 복잡한 programming에는 여러개의 함수가 필요하다. 이때 이러한 함수를 모아서 따로 파일로 저장해 놓고 쓸수 있다.
- 예를 들어 f1, f2라는 함수를 myfunctions.r이라는 파일에 저장했다고 하자.
- 이러한 함수를 R 에 불러와서 사용하려면 다음과 같은 명령어를 사용하면 된다.
- `source("myfunctions.r")`
- 만약 함수를 수정하였다면 위의 `source()` 명령어를 다시 사용하여 update를 해야 한다.

# 효율적인 프로그래밍

- 반복되는 코드를 줄이자. (vectorization, function, etc)
- 계산을 최대한 병렬/분산 처리하자. (apply, parallel, etc)
- 시간이 많이 걸리는 작업 (CPU-intensive part)은 C/C++를 사용할 수 있다.

# Vectorization

상황에 따라 loop 대신 vectorization을 통해 속도를 높일 수 있다.

e.g., 난수 두 개를 더하는 계산을 백만번 하기

```
x <- y <- runif(1000000)
```

```
# looped
```

```
z1 <- c()
```

```
system.time(for (i in 1:1000000) {z1[i] <- x[i] + y[i]}))
```

```
##      user  system elapsed
```

```
##    0.122    0.016    0.138
```

```
# vectorized
```

```
z2 <- c()
```

```
system.time(z2 <- x+y)
```

```
##      user  system elapsed
```

```
##    0.000    0.001    0.001
```

e.g., 십만개의 임의의 수 중 홀수의 개수 세기

*# looped*

```
oddcoun1 <- function(x) {  
  nodd <- 0  
  for (i in seq_along(x)) {  
    if (x[i] %% 2 == 1)  
      nodd <- nodd + 1  
  }  
  return(nodd)  
}
```

*# vectorized*

```
oddcoun2 <- function(x) { return(sum(x %% 2 == 1)) }
```

```
xseq <- sample(1:1000000, 100000, replace = T)
```

```
system.time(oddcount1(xseq))
```

```
##      user  system elapsed
```

```
## 0.011 0.001 0.012
```

```
system.time(oddcount2(xseq))
```

```
##      user  system elapsed
```

```
## 0.001 0.000 0.000
```

# Parallel computing in R

- 병렬 계산을 통해 프로그래밍의 효율을 올릴 수 있음.
- 같은 내용이 수차례 반복되는 시뮬레이션 작업을 여러개로 나눔.
- 자료가 너무 클 때 작게 나누어 분석한 후 결과를 통합함.
- 병렬 계산을 위한 R 패키지: snow, doParallel, foreach, etc



- 1000개의 정규분포 샘플의 평균을 구하는 작업을 5개로 분산하기

```
library(foreach)
library(doParallel)

core_use = detectCores() - 1 # Set the number of cores to use
registerDoParallel(cores = core_use) # Register a number of cores

results = foreach(l = 1:5, .combine = rbind) %dopar% {
  sample = rnorm(1000); mean(sample)
}
results

##           [,1]
## result.1  0.01352558
## result.2 -0.03481708
## result.3 -0.01739682
## result.4 -0.01152818
## result.5 -0.05136705
```

- 100만개의 uniform 분포 샘플의 평균을 구해서 파일로 저장하기

```
mean.parallel <- function(n) {  
  x <- runif(1000000)  
  mx = mean(x)  
  fname = paste('meanx', n, '.RD', sep = '')  
  save(mx, file = fname)  
  return(n)  
}
```

- `mean.parallel()`을 20번 반복 실행하는것을 분산시키기

```
core_use = detectCores() - 1
registerDoParallel(cores = core_use)

results = foreach(l = 1:20, .combine = rbind) %dopar% {
  mean.parallel(l)
}
```

- R object `mx`를 저장한 20개의 파일이 생성되었다.
- 이 파일들에서 `mx`를 불러와 보자

```
xvec = rep(0, 20)
for (i in 1:20) {
  fname = paste('meanx', i, '.RD', sep = '')
  load(fname)
  xvec[i] = mx
}
```

```
xvec
```

```
## [1] 0.5001044 0.5002395 0.4999500 0.4999021 0.5002327 0.5001044
## [8] 0.4995928 0.5001720 0.4997334 0.4998227 0.4997850 0.5001044
## [15] 0.4994657 0.4998816 0.5003901 0.5001044 0.5000224 0.4999500
```

# 정리

- 외부파일을 읽고 쓰는 명령어로 `scan()`, `read.csv()`, `write.csv()`, `write.table()` 등이 있다.
- R object를 저장하고 읽어들이는 명령어로 `save()`, `load()`가 있다.
- `for`, `while`, `if`, `break`, `next`를 잘 이용하여 프로그래밍을 하면 더 간단하게 작업할 수 있다.
- 필요한 함수를 직접 작성하여 사용할 수 있다. R에서는 복잡한 R object도 입력변수와 출력변수로 사용 가능하다.
- vectorization, parallel computing (distributed computing)으로 프로그래밍을 더 효율적으로 할 수 있다.

R을 사용하면서 생기는 여러 문제들, programming을 하면서 생기는 어려움들에 대한 도움을 어디서 구할 수 있을까?

- 온라인 (*StackOverflow*)
- 일부는 오픈소스 커뮤니티에서 R에 사용될 함수들을 개발하고 있다.
- 대부분은 여러분과 비슷하게 온라인 게시판에 질문을 던지고, 그 해결책을 얻고 있다. 해결책은 대부분 R 코드 조각으로 올라온다.
- 프로그래밍을 배우는 가장 좋은 방법은 직접 수행해보는 것.

# 참고자료

- Norman Matloff, The Art of R programming
- 서민구, R을 이용한 데이터 분석 실무 <http://r4pda.co.kr/>
- Jeffrey Stanton, Introduction to Data Science  
<http://jsresearch.net/wiki/projects/teachdatascience>