```r
# Continuation of for loop

# for loop in a matrix - double loop
# We can employ a for loop inside a for loop.

# syntax
# for (var1 in sequence 1) {
#   for (var2 in seqence 2) {
#     statement
#   }
# }

oil.shock <- rnorm(10, 0, 1)         # Suppose that this is a series of oil price from 2000 to 2009
state.response <- runif(5, -1, 1)    # Suppose that this is responses to oil price across 5 states

Economy <- matrix(rep(NA, 50), 10, 5)
rownames(Economy) <- 2000:2009
colnames(Economy) <- state.abb[1:5]

for (i in 1:length(oil.shock)) {
  for (j in 1:length(state.response)) {
    Economy[i,j] <- oil.shock[i]*state.response[j] + rnorm(1)
  }
}

round(Economy,2)

# Alternative way to for loop: matrix operation
# Use the matrix operation to produce the same outcome above (Economy).

oil.shock.mat <- matrix(rep(oil.shock,5),10,5)
state.response.mat <- matrix(rep(state.response,10),10,5,byrow=TRUE)

Economy.1 <- oil.shock.mat * state.response.mat + matrix(rnorm(50),10,5)

round(Economy.1,2)
#####################################################################
# Function

# input => function => return

###############################################################
# use a built in function

?sd
# arguments: X: data (numeric vector), na.rm: logical value.
# Other than default arguments, you should speicify the value

sd(c(1,5,6,7))

value <- c(1,5,6,7)
sd(value)

my_sd <- sd(value)

value <- c(1,5,6,7,NA)
sd(value)

# by position
sd(value, TRUE)
sd(TRUE,value)

# by name
sd(na.rm=TRUE, x=value)

################################################################
# Write a function.

# syntax
# my_fun <- function(arg1, arg2, ...) {
# body
# }

sq <- function(x) {
  square <- x^2
  return(square)
  }

sq(3)
sq(c(3,4,5))
sq(rbind(c(1,2),c(3,4))) # can use different data types.

treatment_efect <- function(tg, cg, trim = FALSE) {
  if (trim) {
    effect <- mean(tg, trim = 0.2) - mean(cg, trim = 0.2)
  } else {
    effect <- mean(tg) - mean(cg)
  }
  variance <- sd(tg)^2 + sd(cg)^2
  t.value <- effect/sqrt(variance)
  if (abs(t.value) > 1.96) {
    print("The average treatment effect is significant")
  } else {
    print("The average treatment effect is not significant")
  }
  result <- list(average_effect=effect, t.value = t.value)
  return(result)
}

# Invoking the function
a <- rnorm(15,5,2)
b <- rnorm(15,0,2)
treatment_efect(a,b)

treatment_efect(tg=a, cg=b, trim=TRUE)
t.value              # variables that are defined in a function are not
                     # accessible outside that function.

source("func.R")     # Read R code from a file

x <- 4
triple(x)

####################################################################
# R packages

# R packages are collections of functions and data sets developed by the community.
# Built in functions such as mean and sd are in the
# base package.
# To use a funciton, You first need to install packages.
# Base package is automatically installed when install R

install.packages("ggplot2")

# load package to the current work session using library() or require()
search()
library(ggplot2)      # you can also use require(ggplot2)
search()
ggplot(mtcars, aes(x = wt, y=mpg)) + geom_point(colour="red") + geom_smooth(method=loess)

ggplot(mtcars, aes(mpg)) + geom_histogram(binwidth=5)

## Exercise 1 #######################################################

setwd("C:/Users/Min Seong Kim/Dropbox/R_programming/lecture/elsect_main")
rev_exp0 <- read.csv("district_rev_exp.csv", na.strings = "-")
```

```r
head(rev_exp0)
str(rev_exp0)

# import "elsect_main.csv" from huskyct.
# calculcate the 20% trimmed mean of "TOTALREV" each state. (use aggregate())

aggregate(x=rev_exp0$TOTALREV, by=list(rev_exp0$STATE), FUN = mean.trim,
          na.rm=TRUE, trim=0.2)

## Exercise 2 ######################################################

# Write a simple function of (x,y) that produce (x+y) - 1/(x+y).
# As you can see, if x+y=0, the outcome is infinity. Instead of
# having infinity, make the return 0.


## Exercise 3 ######################################################

# Using the function and its derivative below, write a function for the
# Newton-Raphson method.

fun <- function(x) {
  y <- x^3 + 2*x + 5
}

fun_der <- function(x) {
  yder <- 3*x^2 + 2
}

curve(fun, xlim=c(-2,2), col='blue', lwd=2, lty=2, ylab='f(x)')

abline(h=0)
abline(v=0)
```