Data Analysis with R - Day 2

ylee@dongguk.edu

2019-4-19

- 코드의 효율을 다루지 않는다.
- 한글 문제는 다루지 않는다.
- 통계학 시간이 아니다.

R as a scientific calculator

```
1 + 2 + 3 + 4 + 5 + 6 + 7 + 8 + 9 + 10

## [1] 55

sqrt(2)

## [1] 1.414214

1.41421^2

## [1] 1.99999

round(1.414214)

## [1] 1

abs(-3.14)

## [1] 3.14

2^5

## [1] 32
```

Any spaces between identifiers seem meaningless, yet putting in proper rhythm, they will improve simplicity, consistency, and aesthetic.

`1 + 2 + 3 + 4 + 5`	`1 + 2+3+4 + 5`
`if (n %% 2 == 1)`	`if( n %%2==1 )`
`lme(distance ~ Sex + age, Orthodont)`	`lme( distance~Sex+age , Orthodont)`

Do not divide any meaningful identifier

```
1 + 2 + 3 + 4 + 5 + 6 +7 + 8 + 9 + 1 0

## Error: <text>:1:38: unexpected numeric constant
## 1: 1 + 2 + 3 + 4 + 5 + 6 +7 + 8 + 9 + 1 0
##                                          ^
```

```
s in(30)

## Error: <text>:1:3: unexpected 'in'
## 1: s in
##      ^
```

```
3 .14 * 64

## Error: <text>:1:3: unexpected numeric constant
## 1: 3 .14
##      ^
```

We can guess what the following functions mean

```r
exp(1)

## [1] 2.718282

log(10)

## [1] 2.302585

sin(30)

## [1] -0.9880316

sqrt(2)

## [1] 1.414214

2^5

## [1] 32
```

exponential function: e^1

```r
exp(1)
```

```
## [1] 2.718282
```

natural logarithmic function: $\log_e 10$

```r
log(10)
```

```
## [1] 2.302585
```

square-root function: $\sqrt{2}$

```r
sqrt(2)
```

```
## [1] 1.414214
```

power operator: $2^5 = 2 \times 2 \times 2 \times 2 \times 2$

```r
2^5
```

```
## [1] 32
```

Basic arithmetic operators[1]

Operators	meaning	examples
`+ x`	$0+x$	`+ 3.14 = +3.14`
`- x`	$0-x$	`- 90 = -90`
`x + y`	$x+y$	`0.5 + 8.1 = 8.6`
`x - y`	$x-y$	`3.14 - 0.02 = 3.12`
`x * y`	$x \times y$	`12 * 30 = 360`
`x / y`	x/y	`35 / 5 = 7`
`x ŷ`	x^y	`4 3̂ = 64`
`x %% y`	modulus of x/y	`7 %% 3 = 1`
`x %/% y`	integer division of x/y	`7 %/% 3 = 2`

[1] "list of R operators": `https://www.statmethods.net/management/operators.html`

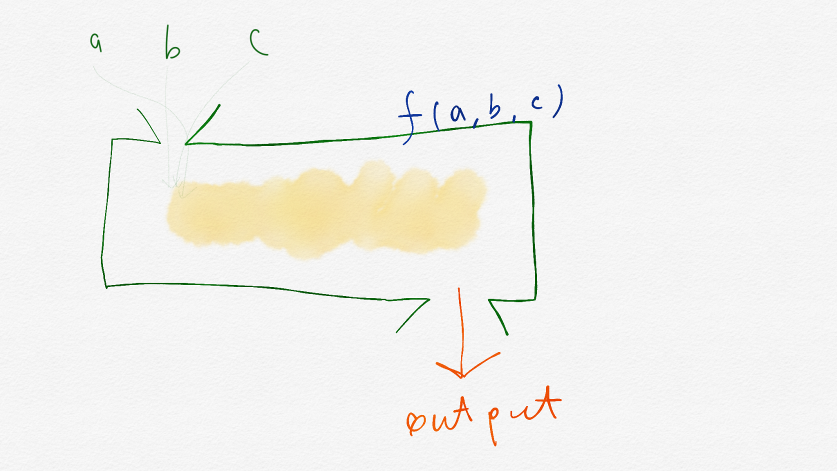
Relational and Logical operators (functions)[2]

Operators	meaning	examples	yield
$x < y$	..	3 < 5	TRUE
$x > y$	..	5 < 3	FALSE
$x <= y$	..	3 <= 3	TRUE
$x >= y$	..	5 >= 7	FALSE
x == y	Does x equal to y?	`32 == 32.1`	FALSE
x != y	Does NOT x equal to y?	`32 != 32.1`	TRUE
! x	Not x	! (pi != 3.14)	TRUE
		! 0	TRUE
x & y	logical AND operator	TRUE & TRUE	TRUE
		1 & 4	TRUE
		4 & 0	FALSE
x | y	logical OR operator	TRUE | FALSE	TRUE
		1 | 0	TRUE
isTRUE (x)	..	isTRUE(5 < 3)	FALSE
isFALSE(x)	..	isFALSE(5 < 3)	TRUE

[2] In logical operation, TRUE corresponds to 1 or any value other than 0 while FALSE only to 0.

Functions

A function receives 0, or 1, or more parameters but always yields a single output.



Actually, R is a function, which many functions comprises, and which every operations are working as functions.

Function	no. of Parameter(s)	performs
log(x)	1	$log_e x$
log(x, base = y)	2	$log_y x$
exp(x)	1	e^x
c(x, y, z, ...)	any greater than 0	concatenate parameters and make them up a single vector
sin(x), cos(x), tan(x)	1	obvious trigonometric functions
load(x)	1	load R dataset named "x"
read.table(x, ...)	1 or many	read text file "x" and make it up to a `data.frame` according to parameters ...
x:y	2	generate a vector starting from x to y by 1
seq(from, to (, by))	2 or 3	generate a vector starting from **from** to **to** (by **by**)
seq(x)	1	generate a vector starting from 1 to x by 1 (or -1)
rep(what, how.long)	2	generate a vector of **how.long**-repetition of **what**. But, when x is a vertor (whose length is greater than 1), the output will be somewhat complex.

Look in Google by "basic R functions list," then you will find `https://cran.r-project.org/doc/contrib/Short-refcard.pdf`

Major statistical functions

Function	no. of Parameter(s)	performs
t.test(x1, x2)	2 vectors in **x1** and **x2**	Welch-Satterthwaite modification of t-test
t.test($y \sim x$)	1 (meaning "y is explained by x")	same as above
t.test(x1, x2, var.equal = TRUE)	3 (2 vectors and 1 flag)	Standard t-test
t.test(x1, x2, paired = TRUE)	3 (2 vectors and 1 flag)	Paired t-test
chisq.test(x)	1 (matrix **x** as a table)	Pearson's X^2-test
aov($y \sim x1 + x2 + \dots$)	1 (meaning "y is explaiend by a series of xs")	analysis of variance (ANOVA) for **y** as dependent variable and **x1**, **x2**, ... as independent variable(s)
anova(m)	1	generate ANOVA table for a model **m**
fisher.test(x)	1 (matrix **x** as a table)	Fisher's exact test
wilcox.test(x1, x2)	2 vectors in **x1** and **x2**	Wilcoxon's rank sum test

For deeper understanding, read built-in manual of functions in R: Try like "**?t.test**" after the R prompt.

Data type

Type	examples	query
Numeric number	0, 4, -3.14, 1.4141	is.numeric(x)
Logical	TRUE, FALSE	is.logical(x)
Character string	"abcde", "Korea", "R programming is easy.", "male/female"	is.character(x)
Factor	inappropriate	is.factor(x)

```r
is.numeric(3.14)
```

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

```r
is.logical(TRUE)
```

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

```r
is.character(754)
```

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

```r
is.character("Tigers are not afraid.")
```

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

```r
is.factor(TRUE)
```

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

```r
is.factor(365)
```

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

```r
is.factor("male")
```

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

Characters are just characters, never be a candidate for arithmetic operations.

```
"Tigers" + "Lions"

## Error in "Tigers" + "Lions": non-numeric argument to binary operator

# paste() function pastes (or concatenates) a series of characters.
paste("Tigers", "Lions")

## [1] "Tigers Lions"

# strsplit() function splits a character into several characters.
strsplit("Tigers are not afraid.", split = " ")

## [[1]]
## [1] "Tigers"  "are"     "not"     "afraid."
```

Factor, one extraordinary atomic data type.

- R factor has a name, called “label.”
- But, It works by its “levels.”

```
gender <- c("M", "F")
is.factor(gender)

## [1] FALSE

is.character(gender)

## [1] TRUE

fGender <- factor(gender)
fGender

## [1] M F
## Levels: F M

is.factor(fGender)

## [1] TRUE
```

No arithmetic/manupulative functions work for R Factors.

Factors are declared names for specific purposes.

```
fGender[1]

## [1] M
## Levels: F M

fGender[1] + fGender[2]

## Warning in Ops.factor(fGender[1], fGender[2]): '+' not meaningful for factors

## [1] NA

# factor-pasting produces a character
p <- paste(fGender[2], fGender[2])
is.factor(p)

## [1] FALSE

is.character(p)

## [1] TRUE
```

Data structure

- Vector
- Matrix
- List
- Data Frame

Vectors

Vectors are the R workhorse[3].

```
x <- 8
x

## [1] 8

x <- c(2, 5, -3, 7)
x

## [1]  2  5 -3  7

length(x)

## [1] 4

y <- c("abc", "de", "f")
y

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

length(y)

## [1] 3

z <- c(1, 3, 5, "k")
typeof(z)

## [1] "character"
```

[3] Norman Matloff in "Art of R Programming"

Indexing a vector

```r
x <- c(2, 5, -3, 7)
x[2]

## [1] 5

y <- c("The", "art", "of", "R", "programming")
length(y)

## [1] 5

y[3]

## [1] "of"

y[c(1, 3)]

## [1] "The" "of"

y[2:4]

## [1] "art" "of"  "R"

(x[2] + x[1]) / x[4]

## [1] 1
```

A matrix is an 2-dimensional vector.

```
x <- c(2, 5, -3, 7)
y <- matrix(x, ncol = 2) # no. of columns
y

##      [,1] [,2]
## [1,]    2   -3
## [2,]    5    7

z <- matrix(1:12, ncol = 4, byrow = TRUE) # fill the matrix by row-first order
z

##      [,1] [,2] [,3] [,4]
## [1,]    1    2    3    4
## [2,]    5    6    7    8
## [3,]    9   10   11   12

x <- c(2, 5, -3, "7") # type of x is coerced to "character"
(y <- matrix(x, ncol = 2))

##      [,1] [,2]
## [1,] "2"  "-3"
## [2,] "5"  "7"

typeof(y)

## [1] "character"
```

Vector operations

```
x <- c(2, 5, -3, 7)
x + 1

## [1]  3  6 -2  8

x / 3

## [1]  0.6666667  1.6666667 -1.0000000  2.3333333

y1 <- seq(0, 360, 10) # from 0 to 360 by 10
y1

##  [1]   0  10  20  30  40  50  60  70  80  90 100 110 120 130 140 150 160
## [18] 170 180 190 200 210 220 230 240 250 260 270 280 290 300 310 320 330
## [35] 340 350 360

y2 <- y1 * pi / 180
```

```r
y2
```

```
##  [1] 0.0000000 0.1745329 0.3490659 0.5235988 0.6981317 0.8726646 1.0471976
##  [8] 1.2217305 1.3962634 1.5707963 1.7453293 1.9198622 2.0943951 2.2689280
## [15] 2.4434610 2.6179939 2.7925268 2.9670597 3.1415927 3.3161256 3.4906585
## [22] 3.6651914 3.8397244 4.0142573 4.1887902 4.3633231 4.5378561 4.7123890
## [29] 4.8869219 5.0614548 5.2359878 5.4105207 5.5850536 5.7595865 5.9341195
## [36] 6.1086524 6.2831853
```

```r
(y3 <- sin(y2))
```

```
##  [1]  0.000000e+00  1.736482e-01  3.420201e-01  5.000000e-01  6.427876e-01
##  [6]  7.660444e-01  8.660254e-01  9.396926e-01  9.848078e-01  1.000000e+00
## [11]  9.848078e-01  9.396926e-01  8.660254e-01  7.660444e-01  6.427876e-01
## [16]  5.000000e-01  3.420201e-01  1.736482e-01  1.224647e-16 -1.736482e-01
## [21] -3.420201e-01 -5.000000e-01 -6.427876e-01 -7.660444e-01 -8.660254e-01
## [26] -9.396926e-01 -9.848078e-01 -1.000000e+00 -9.848078e-01 -9.396926e-01
## [31] -8.660254e-01 -7.660444e-01 -6.427876e-01 -5.000000e-01 -3.420201e-01
## [36] -1.736482e-01 -2.449294e-16
```

Column-Combination of vectors

```
yy <- cbind(y1, y3)  # cbind() binds vectors by columns
head(yy) # function head() displays first 10 values.

##      y1        y3
## [1,]  0 0.0000000
## [2,] 10 0.1736482
## [3,] 20 0.3420201
## [4,] 30 0.5000000
## [5,] 40 0.6427876
## [6,] 50 0.7660444

yy[3, 1]

## y1
## 20

yy[4, 2]

##  y3
## 0.5

is.matrix(yy)

## [1] TRUE

is.matrix(yy[4, 2])

## [1] FALSE
```

Function	no. of Parameter(s)	performs
log(x)	1	$log_e x$
log(x, base = y)	2	$log_y x$
exp(x)	1	e^x
c(x, y, z, ...)	any greater than 0	concatenate parameters and make them up a single vector
sin(x), cos(x), tan(x)	1	obvious trigonometric functions
load(x)	1	load R dataset named "x"
read.table(x, ...)	1 or many	read text file "x" and make it up to a `data.frame` according to parameters ...
x:y	2	generate a vector starting from x to y by 1
seq(from, to (, by))	2 or 3	generate a vector starting from **from** to **to** (by **by**)
seq(x)	1	generate a vector starting from 1 to x by 1 (or -1)
rep(what, how.long)	2	generate a vector of **how.long**-repetition of **what**. But, when x is a vertor (whose length is greater than 1), the output will be somewhat complex.

These 4 functions are automatic vector-generator most commonly used.

x:y generates a vector that is incremental or decremental **by 1**

```
1:5; -1:-5

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

1:-7

## [1]  1  0 -1 -2 -3 -4 -5 -6 -7

1:10 * 100

##  [1]  100  200  300  400  500  600  700  800  900 1000

1:10 / 2; 1:(10 / 2)

##  [1] 0.5 1.0 1.5 2.0 2.5 3.0 3.5 4.0 4.5 5.0
## [1] 1 2 3 4 5

1:9.5

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

0.5:14

##  [1]  0.5  1.5  2.5  3.5  4.5  5.5  6.5  7.5  8.5  9.5 10.5 11.5 12.5 13.5
```

seq() generates a vector having a typical sequence.

```
seq(1, 10) # from 1 to 10 by 1

##  [1]  1  2  3  4  5  6  7  8  9 10

seq(1, 10, 2) # from 1 to 10 by 2

## [1] 1 3 5 7 9

# same result with full inserting full parameter names
seq(from = 1, to = 10, by = 2)

## [1] 1 3 5 7 9

seq(1, 10, length = 3) # from 1 to 10 in a vector whose length is 3.

## [1]  1.0  5.5 10.0

seq(9) # same as 1:9

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

rep() generates a vector having a pattern.

```r
rep(3, 10) # 10-repetition of 3

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

rep(c(3, 5), 10) # 10-repetition of 3, 5

##  [1] 3 5 3 5 3 5 3 5 3 5 3 5 3 5 3 5 3 5 3 5

rep(c(3, 5), each = 10) # 10-repetition of 3 followed by 10-repetition of 5

##  [1] 3 3 3 3 3 3 3 3 3 3 5 5 5 5 5 5 5 5 5 5

rep(rep(c(3, 5), 10), 2) # 2-repetition of rep(c(3, 5), 10)

##  [1] 3 5 3 5 3 5 3 5 3 5 3 5 3 5 3 5 3 5 3 5 3 5 3 5 3 5 3 5 3 5 3 5 3 5 3
## [36] 5 3 5 3 5

rep(c("female", "male"), 6)

##  [1] "female" "male"   "female" "male"   "female" "male"   "female"
##  [8] "male"   "female" "male"   "female" "male"

rep(c("female", "male"), each = 6)

##  [1] "female" "female" "female" "female" "female" "female" "male"
##  [8] "male"   "male"   "male"   "male"   "male"
```

Lists

A list is a container for any values, any different types.

```
x <- list(a = 3, b = "abc", d = TRUE)
x
```

```
## $a
## [1] 3
##
## $b
## [1] "abc"
##
## $d
## [1] TRUE
```

```
typeof(x)
```

```
## [1] "list"
```

```
x1 <- rnorm(20, 92)
x2 <- rnorm(20, 80)
o <- t.test(x1, x2)
typeof(o)
```

```
## [1] "list"
```

```r
print(o)
```

```
##
##  Welch Two Sample t-test
##
## data:  x1 and x2
## t = 40.063, df = 37.28, p-value < 2.2e-16
## alternative hypothesis: true difference in means is not equal to 0
## 95 percent confidence interval:
##  11.61901 12.85653
## sample estimates:
## mean of x mean of y
##  92.28916  80.05139
```

```r
str(o) # DO NOT RUN
```

```r
z
```

```
##      [,1] [,2] [,3] [,4]
## [1,]    1    2    3    4
## [2,]    5    6    7    8
## [3,]    9   10   11   12
```

```r
dim(z)
```

```
## [1] 3 4
```

```r
dimnames(z) <- list(c("r1", "r2", "r3"), c("c1", "c2", "c3", "c4"))
z
```

```
##    c1 c2 c3 c4
## r1  1  2  3  4
## r2  5  6  7  8
## r3  9 10 11 12
```

```r
z[2, 3] == z["r2", "c3"]
```

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

Indexing and extracting components out of a list

```
dbPersonal <- list(math = "A", biology = "B", literature = "A", height = 1.7, weight = 72)
dbPersonal
```

```
## $math
## [1] "A"
##
## $biology
## [1] "B"
##
## $literature
## [1] "A"
##
## $height
## [1] 1.7
##
## $weight
## [1] 72
```

```
dbPersonal$height
```

```
## [1] 1.7
```

```
dbPersonal$weight
```

```
## [1] 72
```

```
dbPersonal$bmi <- dbPersonal$weight / dbPersonal$height ^ 2
round(dbPersonal$bmi, 1)
```

```
## [1] 24.9
```

```
dbPersonal$calcBmi <- function(w, h) round(w / h ^ 2, 2)
dbPersonal
```

```
## $math
## [1] "A"
##
## $biology
## [1] "B"
##
## $literature
## [1] "A"
##
## $height
## [1] 1.7
##
## $weight
## [1] 72
##
## $bmi
## [1] 24.91349
##
## $calcBmi
## function (w, h)
## round(w/h^2, 2)
```

Vector-vector arithmetic

```
a <- c(1, 3, 4, 2, 9, 8)
a

## [1] 1 3 4 2 9 8

a + 2

## [1]  3  5  6  4 11 10

b <- c(1, 2)
a / b

## [1] 1.0 1.5 4.0 1.0 9.0 4.0
```

Data frames

Data frames are a core dataset consisting of grid-type arrangement.

```
Diet <- data.frame(
    subject = 1:16,
        baseline = c(159, 93, 130, 174, 148, 148, 85, 180,
            92, 89, 204, 182, 110, 88, 134, 84),
        final = c(194, 122, 158, 154, 93, 90, 101, 99,
            183, 82, 100, 104, 72, 108, 110, 81))
Diet
```

```
##    subject baseline final
## 1        1      159   194
## 2        2       93   122
## 3        3      130   158
## 4        4      174   154
## 5        5      148    93
## 6        6      148    90
## 7        7       85   101
## 8        8      180    99
## 9        9       92   183
## 10      10       89    82
## 11      11      204   100
## 12      12      182   104
## 13      13      110    72
## 14      14       88   108
## 15      15      134   110
## 16      16       84    81
```

subject	baseline	final
1	159	194
2	93	122
3	130	158
4	174	154
5	148	93
6	148	90
7	85	101
8	180	99
9	92	183
10	89	82
11	204	100
12	182	104
13	110	72
14	88	108
15	134	110
16	84	81

In a sense, data frames are ‘lists’ in R.

```
str(Diet)

## 'data.frame': 16 obs. of  3 variables:
##  $ subject : int  1 2 3 4 5 6 7 8 9 10 ...
##  $ baseline: num  159 93 130 174 148 148 85 180 92 89 ...
##  $ final   : num  194 122 158 154 93 90 101 99 183 82 ...

Diet$subject

##  [1]  1  2  3  4  5  6  7  8  9 10 11 12 13 14 15 16

Diet$baseline[1:4]

## [1] 159  93 130 174

Diet$final - Diet$baseline

##  [1]   35   29   28  -20  -55  -58   16  -81   91   -7 -104  -78  -38   20
## [15]  -24   -3

is.numeric(Diet$subject)

## [1] TRUE

Diet$subject <- factor(Diet$subject)
is.factor(Diet$subject)

## [1] TRUE
```

Conditional statesments

‘if ( )’

if (conditions) {statements} **else** {statements}

```
a <- 2
b <- 5
if (a < b) {
    print("b is greater than a.")
        } else {
        print("b is not greater than a.")
        }
```

```
## [1] "b is greater than a."
```

```r
if (a < b)
    print("b is greater than a.")
# Errors 1
if (a < b)
    print("b is greater than a.")
else
print("b is not greater than a.")
```

```
## Error: <text>:6:1: unexpected 'else'
## 5:    print("b is greater than a.")
## 6: else
##    ^
```

```
# Errors 2
if (a < b) {
    print("b is greater than a.")
}
else
print("b is not greater than a.")

## Error: <text>:5:1: unexpected 'else'
## 4:        }
## 5: else
##    ^
```

```
# Errors 3
if (a < b) {
    print("b is greater than a.")
}
else {
print("b is not greater than a.")
}

## Error: <text>:5:1: unexpected 'else'
## 4:        }
## 5: else
##    ^
```

```
# No errors
if (a < b) {
    print("b is greater than a.")
        } else
        print("b is not greater than a.")
```

```
## [1] "b is greater than a."
```

```
# Errors
if (a < b) print("b is greater than a.")
    else print("b is not greater than a.")


## Error: <text>:3:5: unexpected 'else'
## 2: if (a < b) print("b is greater than a.")
## 3:     else
##         ^
```

```r
# No Errors
if (a < b) print("b is greater than a.") else print("b is not greater than a.")
```

```
## [1] "b is greater than a."
```

X `if (conditions) {statement}`
  `else {statement}`

O `if (conditions) {statement} else {statement}`

청소년(나이 18세 미만)의 입장요금은 7000원이고 노인(만 65세부터)의 입장요금은 7000원, 보통 성인의 입장요금은 15000원인 공원에서 나이를 age 변수에 입력했을 때 입장료를 출력하는 R 구문을 작성하라.

```r
age <- 65
if (age < 18 | age >= 65) print(7000) else print(15000)

## [1] 7000
```

```r
age <- 43
price <- if (age < 18 | age >= 65) 7000 else 15000
price
```

```
## [1] 15000
```

```r
# same output with less intuitive (but stout) approach
price <- ifelse(age < 18, 7000, ifelse(age >= 65, 7000, 15000))
```

Loops

for (assign **indexer** to a **range**)

```
s <- 0
for (i in 1:100) # for () loop runs by order of i = 1, 2, 3, 4, ... 100
        s <- s + i
print(s)

## [1] 5050
```

The R `for ()` assigns how many times a loop runs with what indexer values at the head of the loop.

while (condition for entering a loop)

```
s <- 0
i <- 1
while (i <= 100) {  # while() loop runs only when i <= 100
        s <- s + i
        i <- i + 1
}
print(s)

## [1] 5050
```

The R `while ()` assigns nothing about how many times a loop runs with what indexer values. It only verifies a condition at the first of the loop.

repeat

```
s <- 0
i <- 0
repeat {
        i <- i + 1
        s <- s + i
        if (i == 100) break
}
print(s)

## [1] 5050
```

The R `repeat` assigns nothing about how many times a loop runs with what indexer values. It only repeats the loop. To exit the loop, you must provide an escape statement.

특정 자연수를 입력 받아서 1부터 그 자연수까지의 합을 구하되, 1부터 더해 나가지 않고 거꾸로 더해 나가는 R 코드를 세 가지 반복구문으로(loop) 만들라.

User-defined functions

Functions

A function receives 0, or 1, or more parameters but always yields a single output.

And you can make up a function as you want it to perform.

```
age <- 43
price <- if (age < 18 | age >= 65) 7000 else 15000
price

## [1] 15000

calcPrice <- function(age)
    return(ifelse(age < 18, 7000, ifelse(age >= 65, 7000, 15000)))
age <- c(8, 12, 35, 45, 78)
cbind(age, calcPrice(age))

##      age
## [1,]   8  7000
## [2,]  12  7000
## [3,]  35 15000
## [4,]  45 15000
## [5,]  78  7000

# if age < 0
```

Exercises

심폐우회로를 적용하는 심장수술에서 혈류량을 정할 때 임상의사들은 환자의 체표면적(body surface area)을 기준으로 한다. 체표면적은 측정할 있지만 임상상황에서 사실상 불가능하여 근사식을 이용하여 계산하는데 계산에 필요한 변수는 키와 체중이다. 키를 height 변수에 cm 단위로 넣고, 체중을 weight 변수에 kg 단위로 넣은 후 호출하면 체표면적을 되돌려 주는 함수 calcBSA(height, height)를 작성하라. (Mosteller의 공식을 이용한다)

피보나치 수열은 단순 덧셈으로 이루어졌으면서 자연계에서 여러 응용사례를 들 수 있는 유용한 수열이다. 정의는 단순하다.

1. 수열의 n번째 값은 n-1과 n-2의 합이다.
2. 수열의 0번째 값은 0이고 1번째 값은 1이다.

다음을 수행하고 행 단위로 설명할 수 있다:

수행 1: 0부터 9까지, 즉 첫 10개의 피보나치 수열을 계산해서 벡터 F에 저장한 뒤 출력하는 R 코드를 만든다.

수행 2: 특정 숫자 n이 주어졌을 때 피보나치 수열의 n번째 값을 계산하는(구하는) 함수를 만들어서 함수 이름은 `getFibonacci(n)`으로 이름 붙인다.